



man Pages(3): Library Routines

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900
U.S.A.

Part No: 805-3175-10
October 1998

Copyright 1998 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303-4900 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, SunDocs, Java, the Java Coffee Cup logo, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a).

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 1998 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, Californie 94303-4900 Etats-Unis. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées du système Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, SunDocs, Java, le logo Java Coffee Cup, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REpondre A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.



Contents

PREFACE xliii

Intro(3) 2

a64l(3C) 164

abort(3C) 165

abs(3C) 166

accept(3N) 167

accept(3XN) 169

aclcheck(3) 172

aclsort(3) 174

actomode(3) 175

acltotext(3) 177

acos(3M) 179

acosh(3M) 180

addch(3XC) 182

addchstr(3XC) 184

addnstr(3XC) 186

addnwstr(3XC) 188

addsev(3C) 190

addseverity(3C) 191

add_wch(3XC) 193
add_wchnstr(3XC) 195
aiocancel(3) 197
aio_cancel(3R) 198
aio_error(3R) 200
aio_fsync(3R) 202
aioread(3) 204
aio_read(3R) 207
aio_return(3R) 210
aio_suspend(3R) 212
aiowait(3) 214
aio_write(3R) 216
asin(3M) 219
assert(3C) 220
atan2(3M) 221
atan(3M) 223
atexit(3C) 224
attr_get(3XC) 225
attroff(3XC) 227
au_open(3) 229
au_preselect(3) 231
au_to(3) 233
au_user_mask(3) 236
basename(3C) 238
baudrate(3XC) 239
beep(3XC) 240
ber_decode(3N) 241
ber_encode(3N) 247

bgets(3G) 251
bind(3N) 252
bind(3XN) 254
bkgd(3XC) 257
bkgrnd(3XC) 259
border(3XC) 261
border_set(3XC) 264
bsdmalloc(3X) 267
bsd_signal(3C) 269
bsearch(3C) 270
bstring(3C) 272
btowc(3C) 273
bufsplit(3G) 274
byteorder(3N) 275
cancellation(3T) 276
can_change_color(3XC) 282
catgets(3C) 285
catopen(3C) 286
cbreak(3XC) 289
cbrt(3M) 290
ceil(3M) 291
cfgetispeed(3) 292
cfsetispeed(3) 293
chgat(3XC) 294
cldap_close(3N) 296
cldap_open(3N) 297
cldap_search_s(3N) 298
cldap_setretryinfo(3N) 300

clear(3XC) 301
clearok(3XC) 302
clock(3C) 304
clock_settime(3R) 305
closedir(3C) 307
clrtobot(3XC) 308
clrtoeol(3XC) 309
cond_init(3T) 310
condition(3T) 315
config_admin(3X) 317
config_change_state(3X) 317
confstr(3C) 326
connect(3N) 332
connect(3XN) 335
ConnectToServer(3X) 339
copylist(3G) 340
copysign(3M) 341
copywin(3XC) 342
cos(3M) 344
cosh(3M) 345
crypt(3C) 346
crypt(3X) 347
cset(3C) 349
ctermid(3S) 351
ctime(3C) 352
ctype(3C) 357
curs_addch(3X) 361
curs_addchstr(3X) 364

curs_addstr(3X) 366
curs_addwch(3X) 368
curs_addwchstr(3X) 371
curs_addwstr(3X) 373
curs_alecompat(3X) 374
curs_attr(3X) 376
curs_beep(3X) 378
curs_bkgd(3X) 379
curs_border(3X) 381
curs_clear(3X) 383
curs_color(3X) 385
curs_delch(3X) 388
curs_deleteln(3X) 389
curses(3X) 391
curses(3XC) 407
curs_getch(3X) 420
curs_getstr(3X) 425
curs_getwch(3X) 426
curs_getwstr(3X) 431
curs_getyx(3X) 433
curs_inch(3X) 435
curs_inchstr(3X) 437
curs_initscr(3X) 439
curs_inopts(3X) 441
curs_insch(3X) 445
curs_insstr(3X) 446
curs_instr(3X) 448
curs_inswch(3X) 450

curs_inswstr(3X) 451
curs_inwch(3X) 453
curs_inwchstr(3X) 455
curs_inwstr(3X) 457
curs_kernel(3X) 459
curs_move(3X) 462
curs_outopts(3X) 463
curs_overlay(3X) 466
curs_pad(3X) 468
curs_printw(3X) 470
curs_refresh(3X) 472
curs_scanw(3X) 474
curs_scr_dump(3X) 476
curs_scroll(3X) 478
curs_set(3XC) 479
curs_slk(3X) 480
curs_termattrs(3X) 482
curs_termcap(3X) 484
curs_terminfo(3X) 486
curs_touch(3X) 490
curs_util(3X) 492
curs_window(3X) 494
cuserid(3S) 497
dbm(3B) 498
dbm_clearerr(3) 501
decimal_to_floating(3) 506
def_prog_mode(3XC) 508
delay_output(3XC) 509

delch(3XC) 510
del_curterm(3XC) 511
deleteln(3XC) 513
delscreen(3XC) 514
delwin(3XC) 515
demangle(3) 516
derwin(3XC) 518
des_crypt(3) 520
devid_get(3) 522
dial(3N) 525
di_binding_name(3) 528
di_child_node(3) 531
di_devfs_path(3) 533
difftime(3C) 534
di_init(3) 535
di_minor_devt(3) 539
di_minor_next(3) 541
di_prom_init(3) 542
di_prom_prop_data(3) 543
di_prom_prop_lookup_bytes(3) 545
di_prop_bytes(3) 547
di_prop_lookup_bytes(3) 550
di_prop_next(3) 552
directio(3C) 553
dirname(3C) 556
DisconnectToServer(3X) 558
div(3C) 559
di_walk_minor(3) 560

di_walk_node(3) 562
dladdr(3X) 564
dlclose(3X) 566
dlldump(3X) 567
dlerror(3X) 574
dlinfo(3X) 575
dlopen(3X) 577
dlsym(3X) 582
DmiAddComponent(3X) 584
DmiAddRow(3X) 589
DmiGetConfig(3X) 595
DmiListAttributes(3X) 599
DmiRegisterCi(3X) 606
doconfig(3N) 609
door_bind(3X) 612
door_call(3X) 615
door_create(3X) 618
door_cred(3X) 620
door_info(3X) 621
door_return(3X) 623
door_revoke(3X) 624
door_server_create(3X) 625
doupdate(3XC) 628
drand48(3C) 629
dup2(3C) 632
dupwin(3XC) 633
echo(3XC) 634
echochar(3XC) 635

echo_wchar(3XC) 636
econvert(3) 637
ecvt(3C) 639
elf32_fsize(3E) 641
elf32_getehdr(3E) 642
elf32_getphdr(3E) 644
elf32_getshdr(3E) 646
elf32_xlatetof(3E) 648
elf(3E) 650
elf_begin(3E) 656
elf_cntl(3E) 662
elf_errmsg(3E) 664
elf_fill(3E) 666
elf_flagdata(3E) 667
elf_getarhdr(3E) 669
elf_getarsym(3E) 671
elf_getbase(3E) 672
elf_getdata(3E) 673
elf_getident(3E) 678
elf_getscn(3E) 680
elf_hash(3E) 682
elf_kind(3E) 683
elf_rawfile(3E) 684
elf_strptr(3E) 686
elf_update(3E) 687
elf_version(3E) 691
encrypt(3C) 693
end(3C) 694

endhostent(3XN) 695
endnetent(3XN) 698
endprotoent(3XN) 700
endservent(3XN) 702
endwin(3XC) 704
erasechar(3XC) 705
erf(3M) 706
ethers(3N) 707
euclen(3C) 709
exit(3C) 710
exp(3M) 711
expm1(3M) 712
fabs(3M) 713
fattach(3C) 714
__fbufsize(3S) 716
fclose(3S) 718
fdatasync(3R) 720
fdetach(3C) 721
fdopen(3S) 723
ferror(3S) 725
fflush(3S) 726
ffs(3C) 728
fgetc(3S) 729
fgetpos(3S) 732
fgetwc(3S) 733
filter(3XC) 735
floating_to_decimal(3) 736
flock(3B) 738

flockfile(3S) 740
floor(3M) 742
flushinp(3XC) 743
fmod(3M) 744
fmtmsg(3C) 745
fn_attr_bind(3N) 752
fn_attr_create_subcontext(3N) 754
fn_attr_ext_search(3N) 755
fn_attr_get(3N) 762
fn_attr_get_ids(3N) 764
fn_attr_get_values(3N) 766
FN_attribute_t(3N) 769
fn_attr_modify(3N) 772
FN_attrmodlist_t(3N) 775
fn_attr_multi_get(3N) 778
fn_attr_multi_modify(3N) 782
fn_attr_search(3N) 784
FN_attrset_t(3N) 789
FN_attrvalue_t(3N) 792
FN_composite_name_t(3N) 793
FN_compound_name_t(3N) 798
fn_ctx_bind(3N) 804
fn_ctx_create_subcontext(3N) 806
fn_ctx_destroy_subcontext(3N) 807
fn_ctx_equivalent_name(3N) 809
fn_ctx_get_ref(3N) 811
fn_ctx_get_syntax_attrs(3N) 812
fn_ctx_handle_destroy(3N) 814

fn_ctx_handle_from_initial(3N) 815
fn_ctx_handle_from_ref(3N) 817
fn_ctx_list_bindings(3N) 819
fn_ctx_list_names(3N) 821
fn_ctx_lookup(3N) 824
fn_ctx_lookup_link(3N) 825
fn_ctx_rename(3N) 826
FN_ctx_t(3N) 828
fn_ctx_unbind(3N) 831
FN_identifier_t(3N) 832
fnmatch(3C) 833
FN_ref_addr_t(3N) 835
FN_ref_t(3N) 838
FN_search_control_t(3N) 842
FN_search_filter_t(3N) 845
FN_status_t(3N) 853
FN_string_t(3N) 858
fopen(3B) 862
fopen(3S) 864
form_cursor(3X) 867
form_data(3X) 868
form_driver(3X) 869
form_field(3X) 873
form_field_attributes(3X) 875
form_field_buffer(3X) 877
form_field_info(3X) 879
form_field_just(3X) 881
form_field_new(3X) 883

form_field_opts(3X) 885
form_fieldtype(3X) 887
form_field_userptr(3X) 889
form_field_validation(3X) 890
form_hook(3X) 892
form_new(3X) 894
form_new_page(3X) 895
form_opts(3X) 896
form_page(3X) 898
form_post(3X) 900
forms(3X) 902
form_userptr(3X) 907
form_win(3X) 908
fpgetround(3C) 910
fputc(3S) 912
fputwc(3S) 916
fputws(3S) 919
fread(3S) 920
freopen(3S) 922
frexp(3C) 925
fseek(3S) 926
fsetpos(3S) 929
fsync(3C) 930
ftell(3S) 932
ftime(3C) 933
ftok(3C) 934
ftw(3C) 936
fwide(3C) 939

fwprintf(3S) 940
fwscanf(3S) 947
gelf(3E) 954
gelf_fsize(3E) 954
getacinfo(3) 960
getaclassent(3) 962
getauditflags(3) 965
getauevent(3) 967
getauusernam(3) 970
getbegyx(3XC) 973
getcchar(3XC) 974
getch(3XC) 975
getcwd(3C) 980
getdate(3C) 982
getdtablesize(3C) 989
getenv(3C) 990
getexecname(3C) 991
getfauditflags(3) 992
getgrnam(3C) 994
gethostbyname(3N) 998
gethostid(3C) 1004
gethostname(3C) 1005
gethostname(3XN) 1006
gethrtime(3C) 1007
getloadavg(3C) 1009
getlogin(3C) 1010
getmntent(3C) 1012
getnetbyname(3N) 1015

getnetconfig(3N) 1019
getnetgrent(3N) 1021
getnetpath(3N) 1024
getnstr(3XC) 1026
getn_wstr(3XC) 1028
getopt(3C) 1030
getpagesize(3C) 1033
getpass(3C) 1034
getpeername(3N) 1036
getpeername(3XN) 1037
getpriority(3C) 1039
getprotobyname(3N) 1041
getpublickey(3N) 1045
getpw(3C) 1046
getpwnam(3C) 1047
getrpcbyname(3N) 1052
getrusage(3C) 1056
gets(3S) 1059
getservbyname(3N) 1060
getsockname(3N) 1065
getsockname(3XN) 1066
getsockopt(3N) 1068
getsockopt(3XN) 1072
getspnam(3C) 1075
getsubopt(3C) 1079
gettext(3C) 1082
gettimeofday(3B) 1086
gettimeofday(3C) 1088

gettxt(3C) 1090
getusershell(3C) 1092
getutent(3C) 1093
getutxent(3C) 1096
getvfsent(3C) 1100
getwc(3S) 1102
get_wch(3XC) 1103
getwchar(3S) 1105
getwd(3C) 1106
getwidth(3C) 1107
getwin(3XC) 1108
getws(3S) 1109
glob(3C) 1110
global_variables(3XC) 1115
gmatch(3G) 1116
grantpt(3C) 1117
halfdelay(3XC) 1118
has_ic(3XC) 1119
hline(3XC) 1120
hline_set(3XC) 1122
hsearch(3C) 1124
htonl(3XN) 1127
hypot(3M) 1129
iconv(3) 1130
iconv_close(3) 1135
iconv_open(3) 1136
idcok(3XC) 1138
ilogb(3M) 1139

immedok(3XC) 1140
inch(3XC) 1141
inchnstr(3XC) 1142
index(3C) 1144
inet(3N) 1145
inet_addr(3XN) 1148
initgroups(3C) 1151
initscr(3XC) 1152
innstr(3XC) 1153
innwstr(3XC) 1155
insch(3XC) 1157
insdelln(3XC) 1158
insertln(3XC) 1159
insnstr(3XC) 1160
ins_nwstr(3XC) 1162
insque(3C) 1164
ins_wch(3XC) 1165
intrflush(3XC) 1166
in_wch(3XC) 1167
in_wchnstr(3XC) 1168
isaexec(3C) 1170
isastream(3C) 1172
isatty(3C) 1173
isencrypt(3G) 1174
is_linetouched(3XC) 1175
isnan(3C) 1177
isnan(3M) 1179
iswalph(3C) 1180

iswctype(3C) 1182
j0(3M) 1184
kerberos(3N) 1186
kerberos_rpc(3N) 1191
keyname(3XC) 1194
keypad(3XC) 1195
killpg(3C) 1196
krb_realmofhost(3N) 1197
krb_sendauth(3N) 1200
krb_set_tkt_string(3N) 1204
kstat(3K) 1205
kstat_chain_update(3K) 1212
kstat_lookup(3K) 1213
kstat_open(3K) 1214
kstat_read(3K) 1215
kvm_getu(3K) 1216
kvm_nextproc(3K) 1218
kvm_nlist(3K) 1220
kvm_open(3K) 1221
kvm_read(3K) 1224
lckpwwdf(3C) 1226
ldap(3N) 1227
ldap_abandon(3N) 1238
ldap_add(3N) 1239
ldap_bind(3N) 1241
ldap_cache(3N) 1244
ldap_charset(3N) 1246
ldap_compare(3N) 1248

ldap_control_free(3N) 1250
ldap_delete(3N) 1251
ldap_disptmpl(3N) 1253
ldap_entry2text(3N) 1260
ldap_error(3N) 1264
ldap_first_attribute(3N) 1268
ldap_first_entry(3N) 1270
ldap_first_message(3N) 1272
ldap_friendly(3N) 1274
ldap_get_dn(3N) 1276
ldap_getfilter(3N) 1278
ldap_get_values(3N) 1281
ldap_modify(3N) 1283
ldap_modrdn(3N) 1286
ldap_open(3N) 1288
ldap_parse_result(3N) 1290
ldap_result(3N) 1291
ldap_search(3N) 1293
ldap_searchprefs(3N) 1296
ldap_sort(3N) 1298
ldap_ufn(3N) 1300
ldap_url(3N) 1302
ldexp(3C) 1305
lfmt(3C) 1306
lgamma(3M) 1311
libdevinfo(3) 1313
libthread_db(3T) 1316
libtntctl(3X) 1324

lio_listio(3R) 1329
listen(3N) 1333
listen(3XN) 1334
localeconv(3C) 1336
lockf(3C) 1341
log10(3M) 1345
log1p(3M) 1346
log(3M) 1347
logb(3M) 1348
_longjmp(3C) 1349
longname(3XC) 1350
lsearch(3C) 1351
madvise(3) 1353
maillock(3X) 1355
makecontext(3C) 1357
makedev(3C) 1359
malloc(3C) 1360
malloc(3X) 1363
mapmalloc(3X) 1367
matherr(3M) 1369
mblen(3C) 1376
mbrlen(3C) 1377
mbrtowc(3C) 1379
mbsinit(3C) 1381
mbsrtowcs(3C) 1382
mbstowcs(3C) 1384
mbtowc(3C) 1385
mctl(3B) 1386

media_findname(3X) 1388
media_getattr(3X) 1391
media_getid(3X) 1394
memory(3C) 1396
menu_attributes(3X) 1398
menu_cursor(3X) 1400
menu_driver(3X) 1401
menu_format(3X) 1403
menu_hook(3X) 1405
menu_item_current(3X) 1407
menu_item_name(3X) 1409
menu_item_new(3X) 1410
menu_item_opts(3X) 1412
menu_items(3X) 1414
menu_item_userptr(3X) 1416
menu_item_value(3X) 1417
menu_item_visible(3X) 1419
menu_mark(3X) 1420
menu_new(3X) 1421
menu_opts(3X) 1422
menu_pattern(3X) 1424
menu_post(3X) 1425
menus(3X) 1427
menu_userptr(3X) 1431
menu_win(3X) 1432
meta(3XC) 1434
mkdirp(3G) 1435
mkfifo(3C) 1437

mkstemp(3C) 1438
mktemp(3C) 1439
mktime(3C) 1440
mlock(3C) 1442
mlockall(3C) 1444
modf(3C) 1446
monitor(3C) 1447
move(3XC) 1449
mp(3M) 1450
mq_close(3R) 1452
mq_getattr(3R) 1453
mq_notify(3R) 1455
mq_open(3R) 1457
mq_receive(3R) 1461
mq_send(3R) 1463
mq_setattr(3R) 1465
mq_unlink(3R) 1466
msync(3C) 1467
mtmalloc(3t) 1469
mutex(3T) 1472
mutex_init(3T) 1475
mvmcur(3XC) 1488
mvderwin(3XC) 1489
mvprintw(3XC) 1490
mvscanw(3XC) 1492
mvwin(3XC) 1494
nanosleep(3R) 1495
napms(3XC) 1497

netdir(3N) 1498
newpad(3XC) 1503
nextafter(3M) 1505
nice(3B) 1506
nis_db(3N) 1507
nis_error(3N) 1512
nis_groups(3N) 1514
nis_local_names(3N) 1517
nis_names(3N) 1519
nis_objects(3N) 1526
nis_ping(3N) 1536
nis_server(3N) 1538
nis_subr(3N) 1540
nis_tables(3N) 1543
nl(3XC) 1553
nlist(3B) 1554
nlist(3E) 1555
nl_langinfo(3C) 1556
nlsgetcall(3N) 1557
nlsprovider(3N) 1559
nlsrequest(3N) 1560
nodelay(3XC) 1562
noqiflush(3XC) 1563
NOTE(3X) 1564
notimeout(3XC) 1567
offsetof(3C) 1568
opendir(3C) 1569
overlay(3XC) 1571

p2open(3G) 1574
pam(3) 1576
pam_acct_mgmt(3) 1580
pam_authenticate(3) 1582
pam_chauthtok(3) 1584
pam_getenv(3) 1586
pam_getenvlist(3) 1587
pam_get_user(3) 1588
pam_open_session(3) 1590
pam_putenv(3) 1592
pam_setcred(3) 1594
pam_set_data(3) 1596
pam_set_item(3) 1598
pam_sm(3) 1600
pam_sm_acct_mgmt(3) 1604
pam_sm_authenticate(3) 1606
pam_sm_chauthtok(3) 1608
pam_sm_open_session(3) 1611
pam_sm_setcred(3) 1613
pam_start(3) 1615
pam_strerror(3) 1618
panel_above(3X) 1619
panel_move(3X) 1620
panel_new(3X) 1621
panels(3X) 1622
panel_show(3X) 1624
panel_top(3X) 1625
panel_update(3X) 1626

panel_userptr(3X) 1627
panel_window(3X) 1628
pathfind(3G) 1629
pechochar(3XC) 1631
perror(3C) 1632
pfmt(3C) 1633
plock(3C) 1637
plot(3) 1638
popen(3S) 1642
pow(3M) 1644
printf(3B) 1646
printf(3S) 1652
proc_service(3T) 1662
psignal(3B) 1666
psignal(3C) 1667
ps_lgetregs(3T) 1668
ps_pglobal_lookup(3T) 1670
ps_pread(3T) 1671
ps_pstop(3T) 1672
pthread_atfork(3T) 1674
pthread_attr_getdetachstate(3T) 1676
pthread_attr_getguardsize(3T) 1678
pthread_attr_getinheritsched(3T) 1680
pthread_attr_getschedparam(3T) 1682
pthread_attr_getschedpolicy(3T) 1684
pthread_attr_getscope(3T) 1686
pthread_attr_getstackaddr(3T) 1688
pthread_attr_getstacksize(3T) 1689

pthread_attr_init(3T) 1690
pthread_cancel(3T) 1693
pthread_cleanup_pop(3T) 1694
pthread_cleanup_push(3T) 1695
pthread_condattr_getpshared(3T) 1696
pthread_condattr_init(3T) 1698
pthread_cond_init(3T) 1700
pthread_cond_signal(3T) 1702
pthread_cond_wait(3T) 1704
pthread_create(3T) 1707
pthread_detach(3T) 1711
pthread_equal(3T) 1712
pthread_exit(3T) 1713
pthread_getconcurrency(3T) 1715
pthread_getschedparam(3T) 1717
pthread_getspecific(3T) 1719
pthread_join(3T) 1721
pthread_key_create(3T) 1723
pthread_key_delete(3T) 1725
pthread_kill(3T) 1726
pthread_mutexattr_getprioceiling(3T) 1727
pthread_mutexattr_getprotocol(3T) 1729
pthread_mutexattr_getpshared(3T) 1732
pthread_mutexattr_gettype(3T) 1734
pthread_mutexattr_init(3T) 1737
pthread_mutex_getprioceiling(3T) 1739
pthread_mutex_init(3T) 1741
pthread_mutex_lock(3T) 1743

pthread_once(3T) 1746
pthread_rwlockattr_getpshared(3T) 1747
pthread_rwlockattr_init(3T) 1749
pthread_rwlock_init(3T) 1751
pthread_rwlock_rdlock(3T) 1753
pthread_rwlock_unlock(3T) 1755
pthread_rwlock_wrlock(3T) 1757
pthread_self(3T) 1759
pthread_setcancelstate(3T) 1760
pthread_setcanceltype(3T) 1762
pthread_sigmask(3T) 1764
pthread_testcancel(3T) 1769
ptsname(3C) 1771
putenv(3C) 1772
putp(3XC) 1774
putpwent(3C) 1775
puts(3S) 1776
putspent(3C) 1777
putws(3S) 1778
qsort(3C) 1779
raise(3C) 1781
rand(3B) 1782
rand(3C) 1783
random(3C) 1784
rcmd(3N) 1787
readdir(3B) 1789
readdir(3C) 1791
read_vtoc(3X) 1794

realpath(3C) 1796
reboot(3C) 1798
re_comp(3C) 1799
recv(3N) 1800
recv(3XN) 1803
recvfrom(3XN) 1806
recvmsg(3XN) 1810
redrawwin(3XC) 1814
reg_ci_callback(3X) 1815
regcmp(3C) 1816
regcomp(3C) 1819
regexpr(3G) 1825
remainder(3M) 1828
remove(3C) 1829
resetty(3XC) 1830
resolver(3N) 1831
rewind(3S) 1837
rewinddir(3C) 1838
rexec(3N) 1839
rint(3M) 1841
ripoffline(3XC) 1842
rpc(3N) 1843
rpcbind(3N) 1853
rpcb_getmaps(3N) 1853
rpc_clnt_auth(3N) 1856
rpc_clnt_calls(3N) 1858
rpc_clnt_create(3N) 1862
rpc_control(3N) 1868

rpc_gss_getcred(3N) 1870
rpc_gss_get_error(3N) 1872
rpc_gss_get_mechanisms(3N) 1874
rpc_gss_get_principal_name(3N) 1876
rpc_gss_max_data_length(3N) 1878
rpc_gss_mech_to_oid(3N) 1880
rpc_gss_seccreate(3N) 1882
rpc_gss_set_callback(3N) 1884
rpc_gss_set_defaults(3N) 1886
rpc_gss_set_svc_name(3N) 1887
rpc_rac(3N) 1889
rac_drop(3N) 1889
rpcsec_gss(3N) 1893
rpc_soc(3N) 1899
rpc_svc_calls(3N) 1910
rpc_svc_create(3N) 1915
rpc_svc_err(3N) 1919
rpc_svc_reg(3N) 1921
rpc_xdr(3N) 1923
rstat(3N) 1925
rusers(3N) 1927
rwall(3N) 1929
rwlock(3T) 1930
scalb(3M) 1933
scalbn(3M) 1934
scandir(3B) 1935
scanf(3S) 1937
schedctl_init(3X) 1944

sched_getparam(3R) 1946
sched_get_priority_max(3R) 1947
sched_getscheduler(3R) 1949
sched_rr_get_interval(3R) 1951
sched_setparam(3R) 1952
sched_setscheduler(3R) 1955
sched_yield(3R) 1958
scr_dump(3XC) 1959
sctl(3XC) 1960
secure_rpc(3N) 1961
seekdir(3C) 1965
select(3C) 1966
semaphore(3T) 1970
sem_close(3R) 1974
sem_destroy(3R) 1975
sem_getvalue(3R) 1976
sem_init(3R) 1977
sem_open(3R) 1979
sem_post(3R) 1982
sem_unlink(3R) 1984
sem_wait(3R) 1986
send(3N) 1989
send(3XN) 1991
sendmsg(3XN) 1994
sendto(3XN) 1998
setbuf(3S) 2002
setbuffer(3C) 2004
setcat(3C) 2006

setcchar(3XC) 2007
setjmp(3B) 2008
setjmp(3C) 2012
setkey(3C) 2015
setlabel(3C) 2016
setlocale(3C) 2017
setsockopt(3XN) 2020
set_term(3XC) 2023
shm_open(3R) 2024
shm_unlink(3R) 2027
shutdown(3N) 2028
shutdown(3XN) 2029
sigblock(3B) 2031
sigfpe(3) 2033
siginterrupt(3B) 2036
signal(3B) 2037
signal(3C) 2039
significand(3M) 2041
sigqueue(3R) 2042
sigsetops(3C) 2044
sigstack(3B) 2046
sigstack(3C) 2048
sigvec(3B) 2050
sigwaitinfo(3R) 2055
sin(3M) 2057
sinh(3M) 2058
sleep(3B) 2059
sleep(3C) 2060

slk_attroff(3XC) 2061
socket(3N) 2064
socket(3XN) 2067
socketpair(3N) 2070
socketpair(3XN) 2071
spray(3N) 2074
sqrt(3M) 2076
SSAgentIsAlive(3X) 2077
SSAOidCmp(3X) 2081
SSAStringCpy(3X) 2083
ssignal(3C) 2085
standend(3XC) 2086
stdio(3S) 2087
str2sig(3C) 2092
strccpy(3G) 2093
strcoll(3C) 2095
strerror(3C) 2096
strfind(3G) 2097
strfmon(3C) 2098
strftime(3C) 2103
string(3C) 2108
strcasecmp(3C) 2108
string_to_decimal(3) 2112
strptime(3C) 2115
strsignal(3C) 2119
strtod(3C) 2120
strtol(3C) 2122
strtoul(3C) 2125

strtok(3C) 2127
strxfrm(3C) 2128
swab(3C) 2130
sync_instruction_memory(3C) 2131
syncok(3XC) 2132
syscall(3B) 2133
sysconf(3C) 2134
syslog(3) 2142
system(3) 2147
system(3S) 2148
t_accept(3N) 2149
t_alloc(3N) 2153
tan(3M) 2157
tanh(3M) 2158
t_bind(3N) 2159
tcdrain(3) 2164
tcf_flow(3) 2165
tcf_flush(3) 2167
tcgetattr(3) 2168
tcgetpgrp(3) 2169
tcgetsid(3) 2170
t_close(3N) 2171
t_connect(3N) 2173
tcsendbreak(3) 2177
tcsetattr(3) 2178
tcsetpgrp(3) 2180
tcsetpgrp(3C) 2181
td_init(3T) 2182

td_log(3T) 2183
td_sync_get_info(3T) 2184
td_ta_enable_stats(3T) 2187
td_ta_event_addr(3T) 2189
td_ta_get_nthreads(3T) 2194
td_ta_map_addr2sync(3T) 2195
td_ta_map_id2thr(3T) 2196
td_ta_new(3T) 2198
td_ta_setconcurrency(3T) 2200
td_ta_sync_iter(3T) 2201
td_thr_dbsuspend(3T) 2203
td_thr_getgregs(3T) 2205
td_thr_get_info(3T) 2208
td_thr_lockowner(3T) 2212
td_thr_setprio(3T) 2213
td_thr_setsigpending(3T) 2214
td_thr_sleepinfo(3T) 2216
td_thr_tsd(3T) 2217
td_thr_validate(3T) 2218
tell(3C) 2219
telldir(3C) 2220
termattrs(3XC) 2221
termios(3) 2222
termname(3XC) 2223
t_errno(3N) 2224
t_error(3N) 2226
t_free(3N) 2228
tgetent(3XC) 2230

t_getinfo(3N) 2232
t_getprotaddr(3N) 2236
t_getstate(3N) 2238
thr_create(3T) 2240
threads(3T) 2247
thr_exit(3T) 2255
thr_getconcurrency(3T) 2257
thr_getprio(3T) 2259
thr_join(3T) 2262
thr_keycreate(3T) 2264
thr_kill(3T) 2267
thr_main(3T) 2268
thr_min_stack(3T) 2269
thr_self(3T) 2271
thr_sigsetmask(3T) 2272
thr_stksegment(3T) 2277
thr_suspend(3T) 2278
thr_yield(3T) 2280
tigetflag(3XC) 2281
timer_create(3R) 2283
timer_delete(3R) 2285
timer_settime(3R) 2286
times(3B) 2289
t_listen(3N) 2290
t_look(3N) 2293
tmpfile(3S) 2296
tmpnam(3S) 2297
tnftcl_buffer_alloc(3X) 2299

tnfctl_close(3X) 2302
tnfctl_indirect_open(3X) 2304
tnfctl_internal_open(3X) 2307
tnfctl_kernel_open(3X) 2309
tnfctl_pid_open(3X) 2310
tnfctl_probe_apply(3X) 2317
tnfctl_probe_state_get(3X) 2320
tnfctl_register_funcs(3X) 2325
tnfctl_strerror(3X) 2326
tnfctl_trace_attrs_get(3X) 2327
tnfctl_trace_state_set(3X) 2330
TNF_DECLARE_RECORD(3X) 2333
TNF_PROBE(3X) 2336
TNF_PROBE_0(3X) 2336
tnf_process_disable(3X) 2341
toascii(3C) 2343
_tolower(3C) 2344
tolower(3C) 2345
t_open(3N) 2346
t_optmgmt(3N) 2351
_toupper(3C) 2360
toupper(3C) 2361
towctrans(3C) 2362
tolower(3C) 2363
toupper(3C) 2364
tracing(3X) 2365
t_rcv(3N) 2369
t_rcvconnect(3N) 2372

t_rcvdis(3N) 2375
t_rcvrel(3N) 2378
t_rcvreldata(3N) 2380
t_rcvudata(3N) 2383
t_rcvuderr(3N) 2386
t_rcvv(3N) 2389
t_rcvvudata(3N) 2392
truncate(3C) 2395
tsearch(3C) 2398
t_snd(3N) 2401
t_snddis(3N) 2406
t_sndrel(3N) 2409
t_sndreldata(3N) 2411
t_sndudata(3N) 2414
t_sndv(3N) 2418
t_sndvudata(3N) 2422
t_streerror(3N) 2425
t_sync(3N) 2427
t_sysconf(3N) 2430
ttyname(3C) 2431
ttyslot(3C) 2433
t_unbind(3N) 2434
typeahead(3XC) 2436
ualarm(3C) 2437
unctrl(3XC) 2438
ungetc(3S) 2439
ungetch(3XC) 2440
ungetwc(3S) 2441

unlockpt(3C) 2442
use_env(3XC) 2443
usleep(3C) 2444
vfwprintf(3S) 2445
vidattr(3XC) 2446
vlfmt(3C) 2447
volmgt_acquire(3X) 2449
volmgt_check(3X) 2452
volmgt_feature_enabled(3X) 2454
volmgt_inuse(3X) 2455
volmgt_ownspath(3X) 2456
volmgt_release(3X) 2458
volmgt_root(3X) 2460
volmgt_running(3X) 2461
volmgt_symname(3X) 2462
vpfmt(3C) 2465
vprintf(3S) 2467
vsyslog(3) 2469
wait3(3C) 2471
wait(3B) 2474
watchmalloc(3X) 2480
wrtomb(3C) 2483
wscoll(3C) 2485
wcsftime(3C) 2486
wcsrtombs(3C) 2487
wcsstr(3C) 2489
wcstod(3C) 2490
wcstol(3C) 2492

wcstombs(3C) 2495
wcstoul(3C) 2496
wcstring(3C) 2498
wcswidth(3C) 2503
wcsxfrm(3C) 2504
wctob(3C) 2506
wctomb(3C) 2507
wctrans(3C) 2508
wctype(3C) 2509
wcwidth(3C) 2510
wmemchr(3C) 2511
wmemcmp(3C) 2512
wmemcpy(3C) 2513
wmemmove(3C) 2514
wmemset(3C) 2515
wordexp(3C) 2516
wsprintf(3C) 2520
wsscanf(3C) 2521
wstring(3C) 2522
wunctrl(3XC) 2523
xdr(3N) 2524
xdr_admin(3N) 2527
xdr_complex(3N) 2530
xdr_create(3N) 2533
xfn(3N) 2535
xfn_attributes(3N) 2536
xfn_composite_names(3N) 2540
xfn_compound_names(3N) 2541

| | |
|----------------------|-------------|
| xfn_links(3N) | 2545 |
| xfn_status_codes(3N) | 2548 |
| y0(3M) | 2553 |
| ypclnt(3N) | 2555 |
| yp_update(3N) | 2561 |
| Index | 2562 |

PREFACE

Overview

A man page is provided for both the naive user, and sophisticated user who is familiar with the SunOS operating system and is in need of on-line information. A man page is intended to answer concisely the question “What does it do?” The man pages in general comprise a reference manual. They are not intended to be a tutorial.

The following contains a brief description of each section in the man pages and the information it references:

- Section 1 describes, in alphabetical order, commands available with the operating system.
- Section 1M describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes.
- Section 2 describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value.
- Section 3 describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2 of this volume.
- Section 4 outlines the formats of various files. The C structure declarations for the file formats are given where applicable.
- Section 5 contains miscellaneous documentation such as character set tables.
- Section 6 contains available games and demos.

- Section 7 describes various special files that refer to specific hardware peripherals, and device drivers. STREAMS software drivers, modules and the STREAMS-generic set of system calls are also described.
- Section 9 provides reference information needed to write device drivers in the kernel operating systems environment. It describes two device driver interface specifications: the Device Driver Interface (DDI) and the Driver/Kernel Interface (DKI).
- Section 9E describes the DDI/DKI, DDI-only, and DKI-only entry-point routines a developer may include in a device driver.
- Section 9F describes the kernel functions available for use by device drivers.
- Section 9S describes the data structures used by drivers to share information between the driver and the kernel.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the `intro` pages for more information and detail about each section, and `man(1)` for more information about man pages in general.

NAME This section gives the names of the commands or functions documented, followed by a brief description of what they do.

SYNOPSIS This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full pathname is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[] The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.

. . . Ellipses. Several values may be provided for the previous argument, or the previous argument can be specified multiple times, for example, 'filename...'.
"filename...".

| Separator. Only one of the arguments separated by this character can be specified at time.

{ } Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.

PROTOCOL

This section occurs only in subsection 3R to indicate the protocol description file.

DESCRIPTION

This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES.. Interactive commands, subcommands, requests, macros, functions and such, are described under USAGE.

IOCTL

This section appears on pages in Section 7 only. Only the device class which supplies appropriate parameters to the ioctl (2) system call is called `ioctl` and generates its own heading. `ioctl` calls for a specific device are listed alphabetically (on the man page for that specific device). `ioctl` calls are used for a particular class of devices all of which have an `io` ending, such as `mzio(7D)`

OPTIONS

This lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.

OPERANDS

This section lists the command operands and describes how they affect the actions of the command.

OUTPUT

This section describes the output - standard output, standard error, or output files - generated by the command.

RETURN VALUES

If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or -1, these values are listed in

tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.

ERRORS

On failure, most functions place an error code in the global variable `errno` indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.

USAGE

This section is provided as a guidance on use. This section lists special rules, features and commands that require in-depth explanations. The subsections listed below are used to explain built-in functionality:

- Commands
- Modifiers
- Variables
- Expressions
- Input Grammar

EXAMPLES

This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command line entry and machine response is shown. Whenever an example is given, the prompt is shown as `example%` or if the user must be superuser, `example#`. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS and USAGE sections.

ENVIRONMENT VARIABLES

This section lists any environment variables that the command or function affects, followed by a brief description of the effect.

EXIT STATUS

This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion and

values other than zero for various error conditions.

FILES

This section lists all filenames referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.

ATTRIBUTES

This section lists characteristics of commands, utilities, and device drivers by defining the attribute type and its corresponding value. See `attributes(5)` for more information.

SEE ALSO

This section lists references to other man pages, in-house documentation and outside publications.

DIAGNOSTICS

This section lists diagnostic messages with a brief explanation of the condition causing the error.

WARNINGS

This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.

NOTES

This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.

BUGS

This section describes known bugs and wherever possible, suggests workarounds.

CHAPTER

C Library Functions

| | |
|--------------------|---|
| NAME | Intro - introduction to functions and libraries |
| DESCRIPTION | <p>This section describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2 of this volume. Function declarations can be obtained from the <code>#include</code> files indicated on each page. Certain major collections are identified by a letter after the section number:</p> |
| (3B) | <p>These functions constitute the Source Compatibility (with BSD functions) library. It is implemented as a shared object, <code>libucb.so</code>, and as an archive, <code>libucb.a</code>, but is not automatically linked by the C compilation system. Specify <code>-lucb</code> on the <code>cc</code> command line to link with this library, which is located in the <code>/usr/ucb</code> subdirectory. Header files for this library are located within <code>/usr/ucbinclude</code>.</p> |
| (3C) | <p>These functions, together with those of Section 2 and those marked (3S), constitute the standard C library, <code>libc</code>, which is automatically linked by the C compilation system. The standard C library is implemented as a shared object, <code>libc.so</code>, and as an archive, <code>libc.a</code>. C programs are linked with the shared object version of the standard C library by default. Specify <code>-dn</code> on the <code>cc</code> command line to link with the archive version. See <code>libc(4)</code>, <code>cc(1B)</code> for other overrides, and the "C Compilation System" chapter of the <i>ANSI C Programmer's Guide</i> for a discussion. Some functions behave differently in standard-conforming environments. This behavior is noted on the individual manual pages. See <code>standards(5)</code>.</p> |
| (3E) | <p>These functions constitute the ELF access library, <code>libelf</code>, (Extensible Linking Formats). This library provides the interface for the creation and analyses of "elf" files; executables, objects, and shared objects. <code>libelf</code> is implemented as a shared object, <code>libelf.so</code>, and as an archive, <code>libelf.a</code>, but is not automatically linked by the C compilation system. Specify <code>-lelf</code> on the <code>cc</code> command line to link with this library. See <code>libelf(4)</code>.</p> |
| (3G) | <p>These functions constitute the string pattern-matching & pathname manipulation library, <code>libgen</code>. This library is implemented as an archive, <code>libgen.a</code>, but not as a shared object, and is not automatically linked by the C compilation system. Specify <code>-lgen</code> on the <code>cc</code> command line to link with this library.</p> |
| (3K) | <p>These functions allow access to the kernel's virtual memory library, which is implemented as a shared object, <code>libkvm.so</code>, and as an archive, <code>libkvm.a</code>, but is not automatically linked by the C compilation system. Specify <code>-lkvm</code> on the <code>cc</code> command line to link with this library. See <code>libkvm(4)</code>.</p> |

- (3M) These functions constitute the math library, `libm`. This library is implemented as a shared object, `libm.so`, and as an archive, `libm.a`, but is not automatically linked by the C compilation system. Specify `-lm` on the `cc` command line to link with this library. See `libmp(4)`.
- (3N) These functions constitute the Network Service Library, `libnsl`. It is implemented as a shared object, `libnsl.so`, and as an archive, `libnsl.a`, but is not automatically linked by the C compilation system. Specify `-lnsl` on the `cc` command line to link with this library. See `libnsl(4)`.
- Some of the functions documented in `man3n` incorporate other network libraries, including:
- `libsocket` (see `libsocket(4)`),
 - `libresolv` (see `libresolv(4)`),
 - `librpcsvc` (see `librpcsvc(4)`),
 - `libnisdb` (see `libnisdb(4)`),
 - `librac` (see `librac(4)`),
 - `libxfn` (see `libxfn(4)`), and
 - `libkrb` (see `libkrb(4)`).
- Many base networking functions are also available in the X/Open Networking Interfaces library, `libxnet`. See section (3XN) below for more information on the `libxnet` interfaces.
- Under all circumstances, the use of the Sockets API is recommended over the XTI and TLI APIs. If portability to other XPGV4v2 systems is a requirement, the application must use the `libxnet` interfaces. If portability is not required, the sockets interfaces in `libsocket` and `libnsl` are recommended over those in `libxnet`. Between the XTI and TLI APIs, the XTI interfaces (available with `libxnet`) are recommended over the TLI interfaces (available with `libnsl`).
- (3R) These functions constitute the POSIX.4 Realtime library, `libposix4`. It is implemented only as a shared object, `libposix4.so`, and is not automatically linked by the C compilation system. Specify `-lposix4` on the `cc` command line to link with this library. See `libposix4(4)`.
- (3S) These functions constitute the “standard I/O package” (see `stdio(3S)`). They can be compiled using the the standard C library, `libc`, which is automatically linked by the C compilation system. The standard C library is implemented as a shared object, `libc.so`, and as an archive, `libc.a`. See `libc(4)`.

(3T) These functions constitute the threads libraries, `libpthreads` and `libthread`. These libraries are used for building multithreaded applications. `libpthreads` implements the POSIX (see `standards(5)`) threads interface, whereas `libthread` implements the Solaris threads interface.

Both POSIX threads and Solaris threads can be used within the same application. Their implementations are completely compatible with each other; however, only POSIX threads guarantee portability to other POSIX-conforming environments.

When POSIX and Solaris threads are used in the same application, if there are calls with the same name but different semantics, the POSIX semantic supersedes the Solaris threads semantic. For example, the call to `fork()` will imply the `fork1()` semantic in a program linked with the POSIX threads library, whether or not it is also linked with `-lthread` (Solaris threads).

The `libpthreads` and `libthread` libraries are implemented as shared objects, `libpthreads.so` and `libthread.so`, respectively, but not as archived libraries. `libpthreads` and `libthread` are not automatically linked by the C compilation system. Specify `-lpthreads` or `-lthread` on the `cc` command line to link with these libraries. See `libpthreads(4)` and `libthread(4)`.

The following functions are optional under POSIX and are not supported in the current Solaris release.

```
int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
    int protocol);
int pthread_mutexattr_getprotocol(const pthread_mutexattr_t *attr,
    int *protocol);
int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
    int prioceiling);
int pthread_mutexattr_getprioceiling(const pthread_mutexattr_t *attr,
    int *prioceiling);
```

(3X) Specialized libraries. These functions are contained in libraries including, but not limited to,

- `libadm` (see `libadm(4)`),
- `libbsdmalloc`,
- `libcrypt`,
- `libcurses`,
- `libdl` (see `libdl(4)`),
- `libform`,
- `libmail`,

- `libmalloc`,
- `libmapmalloc` (see `libmapmalloc(4)`),
- `libmenu`, and
- `libpanel`.

(3XC) These functions constitute the X/Open Curses library, located in `/usr/xpg4/lib/libcurses.so.1`. This library provides a set of internationalized functions and macros for creating and modifying input and output to a terminal screen. Included in this library are functions for creating windows, highlighting text, writing to the screen, reading from user input, and moving the cursor. X/Open Curses is designed to optimize screen update activities. The X/Open Curses library conforms fully with Issue 4 of the X/Open Extended Curses specification.

(3XN) These functions constitute X/Open networking interfaces which comply with the X/Open CAE Specification, Networking Services, Issue 4 (September, 1994), and are located in `/usr/lib/libxnet.so.1`. See `libxnet(4)` and `standards(5)` for compilation information.

DEFINITIONS

A character is any bit pattern able to fit into a byte on the machine. *Exception:* in some international languages, a “character” may require more than one byte, and is represented in multi-bytes.

The null character is a character with value 0, conventionally represented in the C language as `\0`. A character array is a sequence of characters. A null-terminated character array (a *string*) is a sequence of characters, the last of which is the null character. The null string is a character array containing only the terminating null character. A null pointer is the value that is obtained by casting 0 into a pointer. C guarantees that this value will not match that of any legitimate pointer, so many functions that return pointers return `NULL` to indicate an error. The macro `NULL` is defined in `<stdio.h>`. Types of the form `size_t` are defined in the appropriate headers.

MT-Level of Libraries

See `attributes(5)` for descriptions of library MT-Levels.

FILES

INCDIR usually `/usr/include`

LIBDIR usually `/usr/ccs/lib`

LIBDIR/`libc.so`

LIBDIR/`libc.a`

LIBDIR/`libgen.a`

LIBDIR/libm.a

LIBDIR/libsfm.sa

/usr/lib/libc.so.1

SEE ALSO

ar(1), cc(1B), ld(1), nm(1), fork(2), intro(2), stdio(3S),
pthread_atfork(3T), libadm(4), libc(4), libelf(4), libdl(4),
libdrb(4), libkvm(4), libmapmalloc(4), libmp(4), libnisdb(4),
libnsl(4), librac(4), libresolv(4), librpcsvc(4), libsocket(4),
libpthread(4), libthread(4), libxfn(4), libxnet(4), attributes (5),
standards(5)

Linker and Libraries Guide

Profiling Tools

ANSI C Programmer's Guide

DIAGNOSTICS

For functions that return floating-point values, error handling varies according to compilation mode. Under the `-xt` (default) option to `cc`, these functions return the conventional values 0, `±HUGE`, or NaN when the function is undefined for the given arguments or when the value is not representable. In the `-Xa` and `-Xc` compilation modes, `±HUGE_VAL` is returned in stead of `±HUGE`. (`HUGE_VAL` and `HUGE` are defined in `math.h` to be infinity and the largest-magnitude single-precision number, respectively.)

**NOTES ON
MULTITHREADED
APPLICATIONS**

When compiling a multithreaded application, either the `_POSIX_C_SOURCE`, `_POSIX_PTHREAD_SEMANTICS`, or `_REENTRANT` flag must be defined on the command line. This enables special definitions for functions only applicable to multithreaded applications. For POSIX.1c-conforming applications, define the `_POSIX_C_SOURCE` flag to be `>= 199506L`:

```
cc [flags] file... -D_POSIX_C_SOURCE=199506L -lpthread
```

For POSIX behavior with the Solaris `fork()` and `fork1()` distinction, compile as follows:

```
cc [flags] file... -D_POSIX_PTHREAD_SEMANTICS -lthread
```

For Solaris threads behavior, compile as follows:

```
cc [flags] file... -D_REENTRANT -lthread
```

When building a singlethreaded application, the above flags should be undefined. This generates a binary that is executable on previous Solaris releases, which do not support multithreading.

Unsafe interfaces should be called only from the main thread to ensure the application's safety.

MT-Safe interfaces are denoted in the `ATTRIBUTES` section of the functions and libraries manual pages (see `attributes(5)`). If a manual page does not state explicitly that an interface is MT-Safe, the user should assume that the interface is unsafe.

REALTIME APPLICATIONS

Be sure to have set the environment variable `LD_BIND_NOW` to a non-null value to enable early binding. Refer to the "When Relocations are Processed" chapter in *Linker and Libraries Guide* for additional information.

NOTES

None of the functions, external variables, or macros should be redefined in the user's programs. Any other name may be redefined without affecting the behavior of other library functions, but such redefinition may conflict with a declaration in an included header.

The headers in `INCDIR` provide function prototypes (function declarations including the types of arguments) for most of the functions listed in this manual. Function prototypes allow the compiler to check for correct usage of these functions in the user's program. The `lint` program checker may also be used and will report discrepancies even if the headers are not included with `#include` statements. Definitions for Sections 2, 3C, and 3S are checked automatically. Other definitions can be included by using the `-l` option to `lint`. (For example, `-lm` includes definitions for `libm`.) Use of `lint` is highly recommended. See the `lint` chapter in *Performance Profiling Tools*.

Users should carefully note the difference between `STREAMS` and *stream*. `STREAMS` is a set of kernel mechanisms that support the development of network services and data communication drivers. It is composed of utility routines, kernel facilities, and a set of data structures. A *stream* is a file with its associated buffering. It is declared to be a pointer to a type `FILE` defined in `<stdio.h>`.

In detailed definitions of components, it is sometimes necessary to refer to symbolic names that are implementation-specific, but which are not necessarily

expected to be accessible to an application program. Many of these symbolic names describe boundary conditions and system limits.

In this section, for readability, these implementation-specific values are given symbolic names. These names always appear enclosed in curly brackets to distinguish them from symbolic names of other implementation-specific constants that are accessible to application programs by headers. These names are not necessarily accessible to an application program through a header, although they may be defined in the documentation for a particular system.

In general, a portable application program should not refer to these symbolic names in its code. For example, an application program would not be expected to test the length of an argument list given to a routine to determine if it was greater than `{ARG_MAX}`.

LIST OF FUNCTIONS

| Name | Description |
|-------------------------------------|--|
| <code>COLOR_PAIR(3XC)</code> | See <code>can_change_color(3XC)</code> |
| <code>ConnectToServer(3X)</code> | connect to a DMI service provider |
| <code>DES_FAILED(3)</code> | See <code>des_crypt(3)</code> |
| <code>DisconnectToServer(3X)</code> | disconnect from a DMI service provider |
| <code>DmiAddComponent(3X)</code> | Management Interface database administration functions |
| <code>DmiAddGroup(3X)</code> | See <code>DmiAddComponent(3X)</code> |
| <code>DmiAddLanguage(3X)</code> | See <code>DmiAddComponent(3X)</code> |
| <code>DmiAddRow(3X)</code> | Management Interface operation functions |
| <code>DmiDeleteComponent(3X)</code> | See <code>DmiAddComponent(3X)</code> |
| <code>DmiDeleteGroup(3X)</code> | See <code>DmiAddComponent(3X)</code> |
| <code>DmiDeleteLanguage(3X)</code> | See <code>DmiAddComponent(3X)</code> |
| <code>DmiDeleteRow(3X)</code> | See <code>DmiAddRow(3X)</code> |
| <code>DmiGetAttribute(3X)</code> | See <code>DmiAddRow(3X)</code> |

| | |
|-------------------------------------|---|
| DmiGetConfig(3X) | Management Interface initialization functions |
| DmiGetMultiple(3X) | See DmiAddRow(3X) |
| DmiGetVersion(3X) | See DmiGetConfig(3X) |
| DmiListAttributes(3X) | Management Interface listing functions |
| DmiListClassNames(3X) | See DmiListAttributes(3X) |
| DmiListComponents(3X) | See DmiListAttributes(3X) |
| DmiListComponentsByClass(3X) | See DmiListAttributes(3X) |
| DmiListGroup(3X) | See DmiListAttributes(3X) |
| DmiListLanguages(3X) | See DmiListAttributes(3X) |
| DmiOriginateEvent(3X) | See DmiRegisterCi(3X) |
| DmiRegister(3X) | See DmiGetConfig(3X) |
| DmiRegisterCi(3X) | Service Provider functions for components |
| DmiSetAttribute(3X) | See DmiAddRow(3X) |
| DmiSetConfig(3X) | See DmiGetConfig(3X) |
| DmiSetMultiple(3X) | See DmiAddRow(3X) |
| DmiUnregisterCi(3X) | See DmiRegisterCi(3X) |
| DmiUnregister(3X) | See DmiGetConfig(3X) |
| FD_CLR(3C) | See select(3C) |
| FD_ISSET(3C) | See select(3C) |
| FD_SET(3C) | See select(3C) |
| FD_ZERO(3C) | See select(3C) |
| FN_attribute_t(3N) | an XFN attribute |
| FN_attrmodlist_t(3N) | a list of attribute modifications |

| | |
|--------------------------------|--|
| FN_attrset_t(3N) | a set of XFN attributes |
| FN_attrvalue_t(3N) | an XFN attribute value |
| FN_bindinglist_t(3N) | See fn_ctx_list_bindings(3N) |
| FN_composite_name_t(3N) | a sequence of component names spanning multiple naming systems |
| FN_compound_name_t(3N) | an XFN compound name |
| FN_ctx_t(3N) | an XFN context |
| FN_ext_searchlist_t(3N) | See fn_attr_ext_search(3N) |
| FN_identifier_t(3N) | an XFN identifier |
| FN_multigetlist_t(3N) | See fn_attr_multi_get(3N) |
| FN_namelist_t(3N) | See fn_ctx_list_names(3N) |
| FN_ref_addr_t(3N) | an address in an XFN reference |
| FN_ref_t(3N) | an XFN reference |
| FN_search_control_t(3N) | options for attribute search |
| FN_search_filter_t(3N) | filter expression for attribute search |
| FN_searchlist_t(3N) | See fn_attr_search(3N) |
| FN_status_t(3N) | an XFN status object |
| FN_string_t(3N) | a character string |
| FN_valuelist_t(3N) | See fn_attr_get_values(3N) |
| Intro(3) | introduction to functions and libraries |
| NOTE(3X) | annotate source code with info for tools |

| | |
|----------------------------|---|
| PAIR_NUMBER(3XC) | See can_change_color(3XC) |
| SSAAgentIsAlive(3X) | Sun Solstice Enterprise Agent registration and communication helper functions |
| SSAGetTrapPort(3X) | See SSAAgentIsAlive(3X) |
| SSAOidCmp(3X) | Sun Solstice Enterprise Agent OID helper functions |
| SSAOidCpy(3X) | See SSAOidCmp(3X) |
| SSAOidDup(3X) | See SSAOidCmp(3X) |
| SSAOidFree(3X) | See SSAOidCmp(3X) |
| SSAOidInit(3X) | See SSAOidCmp(3X) |
| SSAOidNew(3X) | See SSAOidCmp(3X) |
| SSAOidStrToOid(3X) | See SSAOidCmp(3X) |
| SSAOidString(3X) | See SSAOidCmp(3X) |
| SSAOidZero(3X) | See SSAOidCmp(3X) |
| SSARegSubagent(3X) | See SSAAgentIsAlive(3X) |
| SSARegSubtable(3X) | See SSAAgentIsAlive(3X) |
| SSARegSubtree(3X) | See SSAAgentIsAlive(3X) |
| SSASendTrap(3X) | See SSAAgentIsAlive(3X) |
| SSAStringCpy(3X) | Sun Solstice Enterprise Agent string helper functions |
| SSAStringInit(3X) | See SSAStringCpy(3X) |
| SSAStringToChar(3X) | See SSAStringCpy(3X) |
| SSAStringZero(3X) | See SSAStringCpy(3X) |
| SSASubagentOpen(3X) | See SSAAgentIsAlive(3X) |
| TNF_DEBUG(3X) | See TNF_PROBE(3X) |

| | |
|--------------------------------|---|
| TNF_DECLARE_RECORD(3X) | TNF type extension interface for probes |
| TNF_DEFINE_RECORD_1(3X) | See TNF_DECLARE_RECORD(3X) |
| TNF_DEFINE_RECORD_2(3X) | See TNF_DECLARE_RECORD(3X) |
| TNF_DEFINE_RECORD_3(3X) | See TNF_DECLARE_RECORD(3X) |
| TNF_DEFINE_RECORD_4(3X) | See TNF_DECLARE_RECORD(3X) |
| TNF_DEFINE_RECORD_5(3X) | See TNF_DECLARE_RECORD(3X) |
| TNF_PROBE(3X) | probe insertion interface |
| TNF_PROBE_0(3X) | See TNF_PROBE(3X) |
| TNF_PROBE_0_DEBUG(3X) | See TNF_PROBE(3X) |
| TNF_PROBE_1(3X) | See TNF_PROBE(3X) |
| TNF_PROBE_1_DEBUG(3X) | See TNF_PROBE(3X) |
| TNF_PROBE_2(3X) | See TNF_PROBE(3X) |
| TNF_PROBE_2_DEBUG(3X) | See TNF_PROBE(3X) |
| TNF_PROBE_3(3X) | See TNF_PROBE(3X) |
| TNF_PROBE_3_DEBUG(3X) | See TNF_PROBE(3X) |
| TNF_PROBE_4(3X) | See TNF_PROBE(3X) |
| TNF_PROBE_4_DEBUG(3X) | See TNF_PROBE(3X) |
| TNF_PROBE_5(3X) | See TNF_PROBE(3X) |
| TNF_PROBE_5_DEBUG(3X) | See TNF_PROBE(3X) |
| WIFEXITED(3B) | See wait(3B) |
| WIFSIGNALED(3B) | See wait(3B) |

| | |
|------------------------|---|
| WIFSTOPPED(3B) | See wait(3B) |
| _NOTE(3X) | See NOTE(3X) |
| __fbufsize(3S) | interfaces to stdio FILE structure |
| __flbf(3S) | See __fbufsize(3S) |
| __fpending(3S) | See __fbufsize(3S) |
| __fpurge(3S) | See __fbufsize(3S) |
| __freadable(3S) | See __fbufsize(3S) |
| __freading(3S) | See __fbufsize(3S) |
| __fwritable(3S) | See __fbufsize(3S) |
| __fwriting(3S) | See __fbufsize(3S) |
| _edata(3C) | See end(3C) |
| _end(3C) | See end(3C) |
| _etext(3C) | See end(3C) |
| _exithandle(3C) | See exit(3C) |
| _flushlbf(3S) | See __fbufsize(3S) |
| _longjmp(3B) | See setjmp(3B) |
| _longjmp(3C) | non-local goto |
| _setjmp(3B) | See setjmp(3B) |
| _setjmp(3C) | See _longjmp(3C) |
| _tolower(3C) | transliterate upper-case characters to lower-case |
| _toupper(3C) | transliterate lower-case characters to upper-case |
| a64l(3C) | convert between long integer and base-64 ASCII string |

| | |
|-------------------------|--|
| abort(3C) | terminate the process abnormally |
| abs(3C) | return absolute value of integer |
| accept(3N) | accept a connection on a socket |
| accept(3XN) | accept a new connection on a socket |
| aclcheck(3) | check the validity of an ACL |
| aclfrommode(3) | See acltomode(3) |
| aclfromtext(3) | See acltotext(3) |
| aclsort(3) | sort an ACL |
| acltomode(3) | convert an ACL to/from permission bits |
| acltotext(3) | convert an internal representation to/from external representation |
| acos(3M) | arc cosine function |
| acosh(3M) | inverse hyperbolic functions |
| add_wch(3XC) | add a complex character (with rendition) to a window |
| add_wchnstr(3XC) | copy a string of complex characters (with renditions) to a window |
| add_wchstr(3XC) | See add_wchnstr(3XC) |
| addch(3X) | See curs_addch(3X) |
| addch(3XC) | add a character (with rendition) to a window |
| addchnstr(3X) | See curs_addchstr(3X) |
| addchnstr(3XC) | See addchstr(3XC) |

| | |
|------------------------|---|
| addchstr(3X) | See curls_addchstr(3X) |
| addchstr(3XC) | copy a character string (with renditions) to a window |
| addnstr(3X) | See curls_addstr(3X) |
| addnstr(3XC) | add a multi-byte character string (without rendition) to a window |
| addnwstr(3X) | See curls_addwstr(3X) |
| addnwstr(3XC) | add a wide-character string to a window |
| addsev(3C) | define additional severities |
| addseverity(3C) | build a list of severity levels for an application for use with fmtmsg |
| addstr(3X) | See curls_addstr(3X) |
| addstr(3XC) | See addnstr(3XC) |
| addwch(3X) | See curls_addwch(3X) |
| addwchnstr(3X) | See curls_addwchstr(3X) |
| addwchstr(3X) | See curls_addwchstr(3X) |
| addwstr(3X) | See curls_addwstr(3X) |
| addwstr(3XC) | See addnwstr(3XC) |
| adjcurspos(3X) | See curls_alecompat(3X) |
| advance(3G) | See regexpr(3G) |
| aio_cancel(3R) | cancel asynchronous I/O request |
| aio_error(3R) | retrieve errors status for an asynchronous I/O operation |
| aio_fsync(3R) | asynchronous file synchronization |

| | |
|------------------------|---|
| aio_read(3R) | asynchronous read from a file |
| aio_return(3R) | retrieve return status of an asynchronous I/O operation |
| aio_suspend(3R) | wait for asynchronous I/O request |
| aio_write(3R) | asynchronous write to a file |
| aiocancel(3) | cancel an asynchronous operation |
| aioread(3) | read or write asynchronous I/O operations |
| aiowait(3) | wait for completion of asynchronous I/O operation |
| aiowrite(3) | See aioread(3) |
| alloca(3C) | See malloc(3C) |
| alphasort(3B) | See scandir(3B) |
| arc(3) | See plot(3) |
| ascftime(3C) | See strftime(3C) |
| asctime(3C) | See ctime(3C) |
| asctime_r(3C) | See ctime(3C) |
| asin(3M) | arc sine function |
| asinh(3M) | See acosh(3M) |
| assert(3C) | verify program assertion |
| asystem(3) | See system(3) |
| atan(3M) | arc tangent function |
| atan2(3M) | arc tangent function |
| atanh(3M) | See acosh(3M) |

| | |
|------------------------|-------------------------------------|
| atexit(3C) | add program termination routine |
| atof(3C) | See strtod(3C) |
| atoi(3C) | See strtol(3C) |
| atol(3C) | See strtol(3C) |
| atoll(3C) | See strtol(3C) |
| attr_get(3XC) | control window attributes |
| attr_off(3XC) | See attr_get(3XC) |
| attr_on(3XC) | See attr_get(3XC) |
| attr_set(3XC) | See attr_get(3XC) |
| attroff(3X) | See curs_attr(3X) |
| attroff(3XC) | change foreground window attributes |
| attron(3X) | See curs_attr(3X) |
| attron(3XC) | See attroff(3XC) |
| attrset(3X) | See curs_attr(3X) |
| attrset(3XC) | See attroff(3XC) |
| au_close(3) | See au_open(3) |
| au_open(3) | construct and write audit records |
| au_preselect(3) | preselect an audit event |
| au_to(3) | create audit record tokens |
| au_to_arg(3) | See au_to(3) |
| au_to_attr(3) | See au_to(3) |
| au_to_data(3) | See au_to(3) |
| au_to_groups(3) | See au_to(3) |

| | |
|--|-------------------------------------|
| <code>au_to_in_addr(3)</code> | See <code>au_to(3)</code> |
| <code>au_to_ipc(3)</code> | See <code>au_to(3)</code> |
| <code>au_to_ipc_perm(3)</code> | See <code>au_to(3)</code> |
| <code>au_to_iport(3)</code> | See <code>au_to(3)</code> |
| <code>au_to_me(3)</code> | See <code>au_to(3)</code> |
| <code>au_to_opaque(3)</code> | See <code>au_to(3)</code> |
| <code>au_to_path(3)</code> | See <code>au_to(3)</code> |
| <code>au_to_process(3)</code> | See <code>au_to(3)</code> |
| <code>au_to_return(3)</code> | See <code>au_to(3)</code> |
| <code>au_to_socket(3)</code> | See <code>au_to(3)</code> |
| <code>au_to_text(3)</code> | See <code>au_to(3)</code> |
| <code>au_user_mask(3)</code> | get user's binary preselection mask |
| <code>au_write(3)</code> | See <code>au_open(3)</code> |
| <code>auth_destroy(3N)</code> | See <code>rpc_clnt_auth(3N)</code> |
| <code>authdes_create(3N)</code> | See <code>rpc_soc(3N)</code> |
| <code>authdes_getucred(3N)</code> | See <code>secure_rpc(3N)</code> |
| <code>authdes_seccreate(3N)</code> | See <code>secure_rpc(3N)</code> |
| <code>authkerb_getucred(3N)</code> | See <code>kerberos_rpc(3N)</code> |
| <code>authkerb_seccreate(3N)</code> | See <code>kerberos_rpc(3N)</code> |
| <code>authnone_create(3N)</code> | See <code>rpc_clnt_auth(3N)</code> |
| <code>authsys_create(3N)</code> | See <code>rpc_clnt_auth(3N)</code> |
| <code>authsys_create_default(3N)</code> | See <code>rpc_clnt_auth(3N)</code> |
| <code>authunix_create(3N)</code> | See <code>rpc_soc(3N)</code> |
| <code>authunix_create_default(3N)</code> | See <code>rpc_soc(3N)</code> |

| | |
|------------------------------|--|
| basename(3C) | return the last element of a path name |
| baudrate(3X) | See curs_termattrs(3X) |
| baudrate(3XC) | return terminal baud rate |
| bcmp(3C) | See bstring(3C) |
| bcopy(3C) | See bstring(3C) |
| beep(3X) | See curs_beep(3X) |
| beep(3XC) | activate audio-visual alarm |
| ber_alloc(3N) | See ber_encode(3N) |
| ber_alloc_t(3N) | See ber_decode(3N) |
| ber_bvdup(3N) | See ber_decode(3N) |
| ber_bvecfree(3N) | See ber_decode(3N) |
| ber_bvfree(3N) | See ber_decode(3N) |
| ber_decode(3N) | Basic Encoding Rules library decoding functions |
| ber_encode(3N) | simplified Basic Encoding Rules library encoding functions |
| ber_first_element(3N) | See ber_decode(3N) |
| ber_flatten(3N) | See ber_decode(3N) |
| ber_flush(3N) | See ber_encode(3N) |
| ber_free(3N) | See ber_decode(3N) |
| ber_get_bitstring(3N) | See ber_decode(3N) |
| ber_get_boolean(3N) | See ber_decode(3N) |
| ber_get_int(3N) | See ber_decode(3N) |
| ber_get_next(3N) | See ber_decode(3N) |
| ber_get_null(3N) | See ber_decode(3N) |

| | |
|------------------------------------|----------------------------------|
| <code>ber_get_stringa(3N)</code> | See <code>ber_decode(3N)</code> |
| <code>ber_get_stringal(3N)</code> | See <code>ber_decode(3N)</code> |
| <code>ber_get_stringb(3N)</code> | See <code>ber_decode(3N)</code> |
| <code>ber_init(3N)</code> | See <code>ber_decode(3N)</code> |
| <code>ber_next_element(3N)</code> | See <code>ber_decode(3N)</code> |
| <code>ber_peek_tag(3N)</code> | See <code>ber_decode(3N)</code> |
| <code>ber_printf(3N)</code> | See <code>ber_encode(3N)</code> |
| <code>ber_put_bitstring(3N)</code> | See <code>ber_encode(3N)</code> |
| <code>ber_put_boolean(3N)</code> | See <code>ber_encode(3N)</code> |
| <code>ber_put_int(3N)</code> | See <code>ber_encode(3N)</code> |
| <code>ber_put_null(3N)</code> | See <code>ber_encode(3N)</code> |
| <code>ber_put_ostring(3N)</code> | See <code>ber_encode(3N)</code> |
| <code>ber_put_seq(3N)</code> | See <code>ber_encode(3N)</code> |
| <code>ber_put_set(3N)</code> | See <code>ber_encode(3N)</code> |
| <code>ber_put_string(3N)</code> | See <code>ber_encode(3N)</code> |
| <code>ber_scanf(3N)</code> | See <code>ber_decode(3N)</code> |
| <code>ber_skiptag(3N)</code> | See <code>ber_decode(3N)</code> |
| <code>ber_start_seq(3N)</code> | See <code>ber_encode(3N)</code> |
| <code>ber_start_set(3N)</code> | See <code>ber_encode(3N)</code> |
| <code>bgets(3G)</code> | read stream up to next delimiter |
| <code>bind(3N)</code> | bind a name to a socket |
| <code>bind(3XN)</code> | bind a name to a socket |
| <code>bindtextdomain(3C)</code> | See <code>gettext(3C)</code> |
| <code>bkgd(3X)</code> | See <code>curs_bkgd(3X)</code> |

| | |
|-------------------------|---|
| bkgd(3XC) | set the background character (and rendition) of window |
| bkgdset(3X) | See curs_bkgd(3X) |
| bkgdset(3XC) | See bkgd(3XC) |
| bkgrnd(3XC) | set or get the background character (and rendition) of window using a complex character |
| bkgrndset(3XC) | See bkgrnd(3XC) |
| border(3X) | See curs_border(3X) |
| border(3XC) | add a single-byte border to a window |
| border_set(3XC) | use complex characters (and renditions) to draw borders |
| bottom_panel(3X) | See panel_top(3X) |
| box(3) | See plot(3) |
| box(3X) | See curs_border(3X) |
| box(3XC) | See border(3XC) |
| box_set(3XC) | See border_set(3XC) |
| bsd_signal(3C) | simplified signal facilities |
| bsdmalloc(3X) | memory allocator |
| bsearch(3C) | binary search a sorted table |
| bstring(3C) | bit and byte string operations |
| btowc(3C) | single-byte to wide-character conversion |
| bufsplit(3G) | split buffer into fields |
| byteorder(3N) | convert values between host and network byte order |

| | |
|------------------------------|--|
| bzero(3C) | See bstring(3C) |
| calloc(3C) | See malloc(3C) |
| calloc(3X) | See malloc(3X) |
| calloc(3t) | See mtmalloc(3t) |
| callrpc(3N) | See rpc_soc(3N) |
| can_change_color(3X) | See curs_color(3X) |
| can_change_color(3XC) | manipulate color information |
| cancellation(3T) | overview of concepts related to POSIX thread cancellation |
| catclose(3C) | See catopen(3C) |
| catgets(3C) | read a program message |
| catopen(3C) | open/close a message catalog |
| cbc_crypt(3) | See des_crypt(3) |
| cbreak(3X) | See curs_inopts(3X) |
| cbreak(3XC) | set input mode controls |
| cbrt(3M) | cube root function |
| ceil(3M) | ceiling value function |
| cfgetispeed(3) | get input and output baud rate |
| cfgetospeed(3) | See cfgetispeed(3) |
| cfree(3X) | See watchmalloc(3X) |
| cfsetispeed(3) | set input and output baud rate |
| cfsetospeed(3) | See cfsetispeed(3) |
| cftime(3C) | See strftime(3C) |
| chgat(3XC) | change the rendition of characters in a window |
| circle(3) | See plot(3) |

| | |
|---|---|
| <code>cldap_close(3N)</code> | dispose of connectionless LDAP pointer |
| <code>cldap_open(3N)</code> | LDAP connectionless communication preparation |
| <code>cldap_search_s(3N)</code> | connectionless LDAP search |
| <code>cldap_setretryinfo(3N)</code> | set connectionless LDAP request retransmission parameters |
| <code>clear(3X)</code> | See <code>clear(3XC)</code> |
| <code>clear(3XC)</code> | clear a window |
| <code>clearerr(3S)</code> | See <code>ferror(3S)</code> |
| <code>clearok(3X)</code> | See <code>clearok(3XC)</code> |
| <code>clearok(3XC)</code> | set terminal output controls |
| <code>clnt_broadcast(3N)</code> | See <code>rpc_soc(3N)</code> |
| <code>clnt_call(3N)</code> | See <code>rpc_clnt_calls(3N)</code> |
| <code>clnt_control(3N)</code> | See <code>rpc_clnt_create(3N)</code> |
| <code>clnt_create(3N)</code> | See <code>rpc_clnt_create(3N)</code> |
| <code>clnt_create_timed(3N)</code> | See <code>rpc_clnt_create(3N)</code> |
| <code>clnt_create_vers(3N)</code> | See <code>rpc_clnt_create(3N)</code> |
| <code>clnt_create_vers_timed(3N)</code> | See <code>rpc_clnt_create(3N)</code> |
| <code>clnt_destroy(3N)</code> | See <code>rpc_clnt_create(3N)</code> |
| <code>clnt_dg_create(3N)</code> | See <code>rpc_clnt_create(3N)</code> |
| <code>clnt_freeres(3N)</code> | See <code>rpc_clnt_calls(3N)</code> |
| <code>clnt_geterr(3N)</code> | See <code>rpc_clnt_calls(3N)</code> |
| <code>clnt_pcreateerror(3N)</code> | See <code>rpc_clnt_create(3N)</code> |
| <code>clnt_perrno(3N)</code> | See <code>rpc_clnt_calls(3N)</code> |

| | |
|---------------------------------------|--------------------------------------|
| <code>clnt_perror(3N)</code> | See <code>rpc_clnt_calls(3N)</code> |
| <code>clnt_raw_create(3N)</code> | See <code>rpc_clnt_create(3N)</code> |
| <code>clnt_spcreateerror(3N)</code> | See <code>rpc_clnt_create(3N)</code> |
| <code>clnt_sperrno(3N)</code> | See <code>rpc_clnt_calls(3N)</code> |
| <code>clnt_sperror(3N)</code> | See <code>rpc_clnt_calls(3N)</code> |
| <code>clnt_tli_create(3N)</code> | See <code>rpc_clnt_create(3N)</code> |
| <code>clnt_tp_create(3N)</code> | See <code>rpc_clnt_create(3N)</code> |
| <code>clnt_tp_create_timed(3N)</code> | See <code>rpc_clnt_create(3N)</code> |
| <code>clnt_vc_create(3N)</code> | See <code>rpc_clnt_create(3N)</code> |
| <code>clntraw_create(3N)</code> | See <code>rpc_soc(3N)</code> |
| <code>clnttcp_create(3N)</code> | See <code>rpc_soc(3N)</code> |
| <code>clntudp_bufcreate(3N)</code> | See <code>rpc_soc(3N)</code> |
| <code>clntudp_create(3N)</code> | See <code>rpc_soc(3N)</code> |
| <code>clock(3C)</code> | report CPU time used |
| <code>clock_getres(3R)</code> | See <code>clock_gettime(3R)</code> |
| <code>clock_gettime(3R)</code> | See <code>clock_gettime(3R)</code> |
| <code>clock_settime(3R)</code> | high-resolution clock operations |
| <code>closedir(3C)</code> | close a directory stream |
| <code>closelog(3)</code> | See <code>syslog(3)</code> |
| <code>closepl(3)</code> | See <code>plot(3)</code> |
| <code>closevt(3)</code> | See <code>plot(3)</code> |
| <code>clrtoobot(3X)</code> | See <code>curs_clear(3X)</code> |
| <code>clrtoobot(3XC)</code> | clear to the end of a window |
| <code>clrtoeol(3X)</code> | See <code>curs_clear(3X)</code> |
| <code>clrtoeol(3XC)</code> | clear to the end of a line |

| | |
|--------------------------------------|---|
| <code>color_content(3X)</code> | See <code>curs_color(3X)</code> |
| <code>color_content(3XC)</code> | See <code>can_change_color(3XC)</code> |
| <code>color_set(3XC)</code> | See <code>attr_get(3XC)</code> |
| <code>compile(3G)</code> | See <code>regexpr(3G)</code> |
| <code>cond_broadcast(3T)</code> | See <code>cond_init(3T)</code> |
| <code>cond_destroy(3T)</code> | See <code>cond_init(3T)</code> |
| <code>cond_init(3T)</code> | condition variables |
| <code>cond_signal(3T)</code> | See <code>cond_init(3T)</code> |
| <code>cond_timedwait(3T)</code> | See <code>cond_init(3T)</code> |
| <code>cond_wait(3T)</code> | See <code>cond_init(3T)</code> |
| <code>condition(3T)</code> | concepts related to condition variables |
| <code>config_admin(3X)</code> | configuration administration interface |
| <code>config_ap_id_cmp(3X)</code> | See <code>config_admin(3X)</code> |
| <code>config_change_state(3X)</code> | See <code>config_admin(3X)</code> |
| <code>config_list(3X)</code> | See <code>config_admin(3X)</code> |
| <code>config_private_func(3X)</code> | See <code>config_admin(3X)</code> |
| <code>config_stat(3X)</code> | See <code>config_admin(3X)</code> |
| <code>config_strerror(3X)</code> | See <code>config_admin(3X)</code> |
| <code>config_test(3X)</code> | See <code>config_admin(3X)</code> |
| <code>config_unload(3X)</code> | See <code>config_admin(3X)</code> |
| <code>confstr(3C)</code> | get configurable variables |
| <code>connect(3N)</code> | initiate a connection on a socket |
| <code>connect(3XN)</code> | connect a socket |
| <code>cont(3)</code> | See <code>plot(3)</code> |

| | |
|--------------------------|--|
| copylist(3G) | copy a file into memory |
| copysign(3M) | return magnitude of first argument and sign of second argument |
| copywin(3X) | See curs_overlay(3X) |
| copywin(3XC) | overlay or overwrite any portion of window |
| cos(3M) | cosine function |
| cosh(3M) | hyperbolic cosine function |
| cplus_demangle(3) | See demangle(3) |
| crypt(3C) | string encoding function |
| crypt(3X) | password and file encryption functions |
| cset(3C) | get information on EUC codesets |
| csetcol(3C) | See cset(3C) |
| csetlen(3C) | See cset(3C) |
| csetno(3C) | See cset(3C) |
| ctermid(3S) | generate path name for controlling terminal |
| ctermid_r(3S) | See ctermid(3S) |
| ctime(3C) | convert date and time to string |
| ctime_r(3C) | See ctime(3C) |
| ctype(3C) | character handling |
| current_field(3X) | See form_page(3X) |
| current_item(3X) | See menu_item_current(3X) |

| | |
|---------------------------------|--|
| <code>curs_addch(3X)</code> | add a character (with attributes) to a curses window and advance cursor |
| <code>curs_addchstr(3X)</code> | add string of characters and attributes to a curses window |
| <code>curs_addstr(3X)</code> | add a string of characters to a curses window and advance cursor |
| <code>curs_addwch(3X)</code> | add a <code>wchar_t</code> character (with attributes) to a curses window and advance cursor |
| <code>curs_addwchstr(3X)</code> | add string of <code>wchar_t</code> characters (and attributes) to a curses window |
| <code>curs_addwstr(3X)</code> | add a string of <code>wchar_t</code> characters to a curses window and advance cursor |
| <code>curs_alecompat(3X)</code> | these functions are added to ALE curses library for moving the cursor by character. |
| <code>curs_attr(3X)</code> | curses character and window attribute control routines |
| <code>curs_beep(3X)</code> | curses bell and screen flash routines |
| <code>curs_bkgd(3X)</code> | curses window background manipulation routines |
| <code>curs_border(3X)</code> | create curses borders, horizontal and vertical lines |
| <code>curs_clear(3X)</code> | clear all or part of a curses window |
| <code>curs_color(3X)</code> | curses color manipulation routines |
| <code>curs_delch(3X)</code> | delete character under cursor in a curses window |

| | |
|---------------------------------|--|
| <code> curs_deleteln(3X)</code> | delete and insert lines in a curses window |
| <code> curs_getch(3X)</code> | get (or push back) characters from curses terminal keyboard |
| <code> curs_getstr(3X)</code> | get character strings from curses terminal keyboard |
| <code> curs_getwch(3X)</code> | get (or push back) <code>wchar_t</code> characters from curses terminal keyboard |
| <code> curs_getwstr(3X)</code> | get <code>wchar_t</code> character strings from curses terminal keyboard |
| <code> curs_getyx(3X)</code> | get curses cursor and window coordinates |
| <code> curs_inch(3X)</code> | get a character and its attributes from a curses window |
| <code> curs_inchstr(3X)</code> | get a string of characters (and attributes) from a curses window |
| <code> curs_initscr(3X)</code> | curses screen initialization and manipulation routines |
| <code> curs_inopts(3X)</code> | curses terminal input option control routines |
| <code> curs_insch(3X)</code> | insert a character before the character under the cursor in a curses window |
| <code> curs_insstr(3X)</code> | insert string before character under the cursor in a curses window |
| <code> curs_instr(3X)</code> | get a string of characters from a curses window |
| <code> curs_inswch(3X)</code> | insert a <code>wchar_t</code> character before the character under the cursor in a curses window |

| | |
|--------------------------------|---|
| <code>curs_inswstr(3X)</code> | insert <code>wchar_t</code> string before character under the cursor in a curses window |
| <code>curs_inwch(3X)</code> | get a <code>wchar_t</code> character and its attributes from a curses window |
| <code>curs_inwchstr(3X)</code> | get a string of <code>wchar_t</code> characters (and attributes) from a curses window |
| <code>curs_inwstr(3X)</code> | get a string of <code>wchar_t</code> characters from a curses window |
| <code>curs_kernel(3X)</code> | low-level curses routines |
| <code>curs_move(3X)</code> | move curses window cursor |
| <code>curs_outopts(3X)</code> | curses terminal output option control routines |
| <code>curs_overlay(3X)</code> | overlap and manipulate overlapped curses windows |
| <code>curs_pad(3X)</code> | create and display curses pads |
| <code>curs_printw(3X)</code> | print formatted output in curses windows |
| <code>curs_refresh(3X)</code> | refresh curses windows and lines |
| <code>curs_scanw(3X)</code> | convert formatted input from a curses widow |
| <code>curs_scr_dump(3X)</code> | read (write) a curses screen from (to) a file |
| <code>curs_scroll(3X)</code> | scroll a curses window |
| <code>curs_set(3X)</code> | See <code>curs_kernel(3X)</code> |
| <code>curs_set(3XC)</code> | set visibility of cursor |
| <code>curs_slk(3X)</code> | curses soft label routines |

| | |
|-----------------------------------|---|
| <code>curs_termattrs(3X)</code> | curses environment query routines |
| <code>curs_termcap(3X)</code> | curses interfaces (emulated) to the termcap library |
| <code>curs_terminfo(3X)</code> | curses interfaces to terminfo database |
| <code>curs_touch(3X)</code> | curses refresh control routines |
| <code>curs_util(3X)</code> | curses miscellaneous utility routines |
| <code>curs_window(3X)</code> | create curses windows |
| <code>curses(3X)</code> | CRT screen handling and optimization package |
| <code>curses(3XC)</code> | introduction and overview of X/Open Curses |
| <code>cuserid(3S)</code> | get character login name of the user |
| <code>data_ahead(3X)</code> | See <code>form_data(3X)</code> |
| <code>data_behind(3X)</code> | See <code>form_data(3X)</code> |
| <code>db_add_entry(3N)</code> | See <code>nis_db(3N)</code> |
| <code>db_checkpoint(3N)</code> | See <code>nis_db(3N)</code> |
| <code>db_create_table(3N)</code> | See <code>nis_db(3N)</code> |
| <code>db_destroy_table(3N)</code> | See <code>nis_db(3N)</code> |
| <code>db_first_entry(3N)</code> | See <code>nis_db(3N)</code> |
| <code>db_free_result(3N)</code> | See <code>nis_db(3N)</code> |
| <code>db_initialize(3N)</code> | See <code>nis_db(3N)</code> |
| <code>db_list_entries(3N)</code> | See <code>nis_db(3N)</code> |
| <code>db_next_entry(3N)</code> | See <code>nis_db(3N)</code> |
| <code>db_remove_entry(3N)</code> | See <code>nis_db(3N)</code> |

| | |
|--------------------------------------|---|
| <code>db_reset_next_entry(3N)</code> | See <code>nis_db(3N)</code> |
| <code>db_standby(3N)</code> | See <code>nis_db(3N)</code> |
| <code>db_table_exists(3N)</code> | See <code>nis_db(3N)</code> |
| <code>db_unload_table(3N)</code> | See <code>nis_db(3N)</code> |
| <code>dbm(3B)</code> | data base subroutines |
| <code>dbm_clearerr(3)</code> | database functions |
| <code>dbm_close(3)</code> | See <code>dbm_clearerr(3)</code> |
| <code>dbm_delete(3)</code> | See <code>dbm_clearerr(3)</code> |
| <code>dbm_error(3)</code> | See <code>dbm_clearerr(3)</code> |
| <code>dbm_fetch(3)</code> | See <code>dbm_clearerr(3)</code> |
| <code>dbm_firstkey(3)</code> | See <code>dbm_clearerr(3)</code> |
| <code>dbm_nextkey(3)</code> | See <code>dbm_clearerr(3)</code> |
| <code>dbm_open(3)</code> | See <code>dbm_clearerr(3)</code> |
| <code>dbm_store(3)</code> | See <code>dbm_clearerr(3)</code> |
| <code>dbmclose(3B)</code> | See <code>dbm(3B)</code> |
| <code>dbminit(3B)</code> | See <code>dbm(3B)</code> |
| <code>dcgettext(3C)</code> | See <code>gettext(3C)</code> |
| <code>decimal_to_double(3)</code> | See <code>decimal_to_floating(3)</code> |
| <code>decimal_to_extended(3)</code> | See <code>decimal_to_floating(3)</code> |
| <code>decimal_to_floating(3)</code> | convert decimal record to floating-point value |
| <code>decimal_to_quadruple(3)</code> | See <code>decimal_to_floating(3)</code> |
| <code>decimal_to_single(3)</code> | See <code>decimal_to_floating(3)</code> |

| | |
|----------------------------------|--|
| <code>def_prog_mode(3X)</code> | See <code>curs_kernel(3X)</code> |
| <code>def_prog_mode(3XC)</code> | save/restore terminal modes |
| <code>def_shell_mode(3X)</code> | See <code>curs_kernel(3X)</code> |
| <code>def_shell_mode(3XC)</code> | See <code>def_prog_mode(3XC)</code> |
| <code>del_curterm(3X)</code> | See <code>curs_terminfo(3X)</code> |
| <code>del_curterm(3XC)</code> | free space pointed to by terminal |
| <code>del_panel(3X)</code> | See <code>panel_new(3X)</code> |
| <code>delay_output(3X)</code> | See <code>curs_util(3X)</code> |
| <code>delay_output(3XC)</code> | delays output |
| <code>delch(3X)</code> | See <code>curs_delch(3X)</code> |
| <code>delch(3XC)</code> | remove a character |
| <code>delete(3B)</code> | See <code>dbm(3B)</code> |
| <code>deleteln(3X)</code> | See <code>curs_deleteln(3X)</code> |
| <code>deleteln(3XC)</code> | remove a line |
| <code>delscreen(3X)</code> | See <code>curs_initscr(3X)</code> |
| <code>delscreen(3XC)</code> | free space associated with the SCREEN data structure |
| <code>delwin(3X)</code> | See <code>curs_window(3X)</code> |
| <code>delwin(3XC)</code> | delete a window |
| <code>demangle(3)</code> | decode a C++ encoded symbol name |
| <code>derwin(3X)</code> | See <code>curs_window(3X)</code> |
| <code>derwin(3XC)</code> | create a new window or subwindow |
| <code>des_crypt(3)</code> | fast DES encryption |
| <code>des_setparity(3)</code> | See <code>des_crypt(3)</code> |

| | |
|--|---|
| <code>devid_compare(3)</code> | See <code>devid_get(3)</code> |
| <code>devid_deviceid_to_nmlist(3)</code> | See <code>devid_get(3)</code> |
| <code>devid_free(3)</code> | See <code>devid_get(3)</code> |
| <code>devid_free_nmlist(3)</code> | See <code>devid_get(3)</code> |
| <code>devid_get(3)</code> | device id interfaces for user applications |
| <code>devid_get_minor_name(3)</code> | See <code>devid_get(3)</code> |
| <code>devid_sizeof(3)</code> | See <code>devid_get(3)</code> |
| <code>dgettext(3C)</code> | See <code>gettext(3C)</code> |
| <code>di_binding_name(3)</code> | return libdevinfo node information |
| <code>di_bus_addr(3)</code> | See <code>di_binding_name(3)</code> |
| <code>di_child_node(3)</code> | libdevinfo node traversal functions |
| <code>di_compatible_names(3)</code> | See <code>di_binding_name(3)</code> |
| <code>di_devfs_path(3)</code> | generate and free physical path names |
| <code>di_devfs_path_free(3)</code> | See <code>di_devfs_path(3)</code> |
| <code>di_devid(3)</code> | See <code>di_binding_name(3)</code> |
| <code>di_driver_name(3)</code> | See <code>di_binding_name(3)</code> |
| <code>di_driver_ops(3)</code> | See <code>di_binding_name(3)</code> |
| <code>di_drv_first_node(3)</code> | See <code>di_child_node(3)</code> |
| <code>di_drv_next_node(3)</code> | See <code>di_child_node(3)</code> |
| <code>di_fini(3)</code> | See <code>di_init(3)</code> |
| <code>di_init(3)</code> | create and destroy a snapshot of kernel device tree |
| <code>di_instance(3)</code> | See <code>di_binding_name(3)</code> |

| | |
|---|--|
| <code>di_minor_devt(3)</code> | return libdevinfo minor node information |
| <code>di_minor_name(3)</code> | See <code>di_minor_devt(3)</code> |
| <code>di_minor_next(3)</code> | libdevinfo minor node tra |
| <code>di_minor_nodetype(3)</code> | See <code>di_minor_devt(3)</code> |
| <code>di_minor_spectype(3)</code> | See <code>di_minor_devt(3)</code> |
| <code>di_node_name(3)</code> | See <code>di_binding_name(3)</code> |
| <code>di_nodeid(3)</code> | See <code>di_binding_name(3)</code> |
| <code>di_parent_node(3)</code> | See <code>di_child_node(3)</code> |
| <code>di_prom_fini(3)</code> | See <code>di_prom_init(3)</code> |
| <code>di_prom_init(3)</code> | create and destroy a handle to the PROM device information |
| <code>di_prom_prop_data(3)</code> | access PROM device information |
| <code>di_prom_prop_lookup_bytes(3)</code> | search for a PROM property |
| <code>di_prom_prop_lookup_ints(3)</code> | See <code>di_prom_prop_lookup_bytes(3)</code> |
| <code>di_prom_prop_lookup_strings(3)</code> | See <code>di_prom_prop_lookup_bytes(3)</code> |
| <code>di_prom_prop_name(3)</code> | See <code>di_prom_prop_data(3)</code> |
| <code>di_prom_prop_next(3)</code> | See <code>di_prom_prop_data(3)</code> |
| <code>di_prop_bytes(3)</code> | access property values and attributes |
| <code>di_prop_devt(3)</code> | See <code>di_prop_bytes(3)</code> |
| <code>di_prop_ints(3)</code> | See <code>di_prop_bytes(3)</code> |
| <code>di_prop_lookup_bytes(3)</code> | search for a property |
| <code>di_prop_lookup_ints(3)</code> | See <code>di_prop_lookup_bytes(3)</code> |

| | |
|--|--|
| <code>di_prop_lookup_strings(3)</code> | See <code>di_prop_lookup_bytes(3)</code> |
| <code>di_prop_name(3)</code> | See <code>di_prop_bytes(3)</code> |
| <code>di_prop_next(3)</code> | libdevinfo property traversal function |
| <code>di_prop_strings(3)</code> | See <code>di_prop_bytes(3)</code> |
| <code>di_prop_type(3)</code> | See <code>di_prop_bytes(3)</code> |
| <code>di_sibling_node(3)</code> | See <code>di_child_node(3)</code> |
| <code>di_walk_minor(3)</code> | traverse libdevinfo minor nodes |
| <code>di_walk_node(3)</code> | traverse libdevinfo device nodes |
| <code>dial(3N)</code> | establish an outgoing terminal line connection |
| <code>difftime(3C)</code> | computes the difference between two calendar times |
| <code>directio(3C)</code> | provide advice to file system |
| <code>dirname(3C)</code> | report the parent directory name of a file path name |
| <code>div(3C)</code> | compute the quotient and remainder |
| <code>dladdr(3X)</code> | translate address to symbolic information |
| <code>dlclose(3X)</code> | close a shared object |
| <code>dldump(3X)</code> | create a new file from a dynamic object component of the calling process |
| <code>dlerror(3X)</code> | get diagnostic information |
| <code>dlinfo(3X)</code> | dynamic load information |
| <code>dlmopen(3X)</code> | See <code>dlopen(3X)</code> |

| | |
|-------------------------------------|---|
| <code>dlopen(3X)</code> | gain access to an executable object file |
| <code>dlsym(3X)</code> | get the address of a symbol in a shared object |
| <code>dn_comp(3N)</code> | See <code>resolver(3N)</code> |
| <code>dn_expand(3N)</code> | See <code>resolver(3N)</code> |
| <code>doconfig(3N)</code> | execute a configuration script |
| <code>door_bind(3X)</code> | bind or unbind the current thread with the door server pool |
| <code>door_call(3X)</code> | invoke the function associated with a door descriptor |
| <code>door_create(3X)</code> | create a door descriptor |
| <code>door_cred(3X)</code> | return credential information associated with the client |
| <code>door_info(3X)</code> | return information associated with a door descriptor |
| <code>door_return(3X)</code> | return from a door invocation |
| <code>door_revoke(3X)</code> | revoke access to a door descriptor |
| <code>door_server_create(3X)</code> | specify an alternative door server thread creation function |
| <code>door_unbind(3X)</code> | See <code>door_bind(3X)</code> |
| <code>double_to_decimal(3)</code> | See <code>floating_to_decimal(3)</code> |
| <code>doupdate(3X)</code> | See <code>curs_refresh(3X)</code> |
| <code>doupdate(3XC)</code> | refresh windows and lines |
| <code>drand48(3C)</code> | generate uniformly distributed pseudo-random numbers |

| | |
|-------------------------------------|--|
| <code>dup2(3C)</code> | duplicate an open file descriptor |
| <code>dup_field(3X)</code> | See <code>form_field_new(3X)</code> |
| <code>dupwin(3X)</code> | See <code>curls_window(3X)</code> |
| <code>dupwin(3XC)</code> | duplicate a window |
| <code>dynamic_field_info(3X)</code> | See <code>form_field_info(3X)</code> |
| <code>ecb_crypt(3)</code> | See <code>des_crypt(3)</code> |
| <code>echo(3X)</code> | See <code>curls_inopts(3X)</code> |
| <code>echo(3XC)</code> | enable/disable terminal echo |
| <code>echo_wchar(3XC)</code> | add a complex character and refresh window |
| <code>echochar(3X)</code> | See <code>curls_addch(3X)</code> |
| <code>echochar(3XC)</code> | add a single-byte character and refresh window |
| <code>echowchar(3X)</code> | See <code>curls_addwch(3X)</code> |
| <code>econvert(3)</code> | output conversion |
| <code>ecvt(3)</code> | See <code>econvert(3)</code> |
| <code>ecvt(3C)</code> | convert floating-point number to string |
| <code>edata(3C)</code> | See <code>end(3C)</code> |
| <code>elf(3E)</code> | object file access library |
| <code>elf32_fsize(3E)</code> | return the size of an object file type |
| <code>elf32_getehdr(3E)</code> | retrieve class-dependent object file header |
| <code>elf32_getphdr(3E)</code> | retrieve class-dependent program header table |
| <code>elf32_getshdr(3E)</code> | retrieve class-dependent section header |

| | |
|---------------------------------|-------------------------------------|
| <code>elf32_newehdr(3E)</code> | See <code>elf32_getehdr(3E)</code> |
| <code>elf32_newphdr(3E)</code> | See <code>elf32_getphdr(3E)</code> |
| <code>elf32_xlatetof(3E)</code> | class-dependent data translation |
| <code>elf32_xlatetom(3E)</code> | See <code>elf32_xlatetof(3E)</code> |
| <code>elf64_fsize(3E)</code> | See <code>elf32_fsize(3E)</code> |
| <code>elf64_getehdr(3E)</code> | See <code>elf32_getehdr(3E)</code> |
| <code>elf64_getphdr(3E)</code> | See <code>elf32_getphdr(3E)</code> |
| <code>elf64_getshdr(3E)</code> | See <code>elf32_getshdr(3E)</code> |
| <code>elf64_newehdr(3E)</code> | See <code>elf32_getehdr(3E)</code> |
| <code>elf64_newphdr(3E)</code> | See <code>elf32_getphdr(3E)</code> |
| <code>elf64_xlatetof(3E)</code> | See <code>elf32_xlatetof(3E)</code> |
| <code>elf64_xlatetom(3E)</code> | See <code>elf32_xlatetof(3E)</code> |
| <code>elf_begin(3E)</code> | process ELF object files |
| <code>elf_cntl(3E)</code> | control an elf file descriptor |
| <code>elf_end(3E)</code> | See <code>elf_begin(3E)</code> |
| <code>elf_errmsg(3E)</code> | error handling |
| <code>elf_errno(3E)</code> | See <code>elf_errmsg(3E)</code> |
| <code>elf_fill(3E)</code> | set fill byte |
| <code>elf_flagdata(3E)</code> | manipulate flags |
| <code>elf_flagehdr(3E)</code> | See <code>elf_flagdata(3E)</code> |
| <code>elf_flagelf(3E)</code> | See <code>elf_flagdata(3E)</code> |
| <code>elf_flagphdr(3E)</code> | See <code>elf_flagdata(3E)</code> |
| <code>elf_flagscn(3E)</code> | See <code>elf_flagdata(3E)</code> |
| <code>elf_flagshdr(3E)</code> | See <code>elf_flagdata(3E)</code> |
| <code>elf_getarhdr(3E)</code> | retrieve archive member header |

| | |
|-------------------------------|---|
| <code>elf_getarsym(3E)</code> | retrieve archive symbol table |
| <code>elf_getbase(3E)</code> | get the base offset for an object file |
| <code>elf_getdata(3E)</code> | get section data |
| <code>elf_getident(3E)</code> | retrieve file identification data |
| <code>elf_getscn(3E)</code> | get section information |
| <code>elf_hash(3E)</code> | compute hash value |
| <code>elf_kind(3E)</code> | determine file type |
| <code>elf_memory(3E)</code> | See <code>elf_begin(3E)</code> |
| <code>elf_ndxscn(3E)</code> | See <code>elf_getscn(3E)</code> |
| <code>elf_newdata(3E)</code> | See <code>elf_getdata(3E)</code> |
| <code>elf_newscn(3E)</code> | See <code>elf_getscn(3E)</code> |
| <code>elf_next(3E)</code> | See <code>elf_begin(3E)</code> |
| <code>elf_nextscn(3E)</code> | See <code>elf_getscn(3E)</code> |
| <code>elf_rand(3E)</code> | See <code>elf_begin(3E)</code> |
| <code>elf_rawdata(3E)</code> | See <code>elf_getdata(3E)</code> |
| <code>elf_rawfile(3E)</code> | retrieve uninterpreted file contents |
| <code>elf_strptr(3E)</code> | make a string pointer |
| <code>elf_update(3E)</code> | update an ELF descriptor |
| <code>elf_version(3E)</code> | coordinate ELF library and application versions |
| <code>encrypt(3C)</code> | encoding function |
| <code>end(3C)</code> | last locations in program |
| <code>endac(3)</code> | See <code>getacinfo(3)</code> |
| <code>endauclass(3)</code> | See <code>getauclassent(3)</code> |

| | |
|-------------------------|--------------------------------------|
| endauevent(3) | See getauevent(3) |
| endauser(3) | See getausernam(3) |
| endgrent(3C) | See getgrnam(3C) |
| endhostent(3N) | See gethostbyname(3N) |
| endhostent(3XN) | network host database functions |
| endnetconfig(3N) | See getnetconfig(3N) |
| endnetent(3N) | See getnetbyname(3N) |
| endnetent(3XN) | network database functions |
| endnetgrent(3N) | See getnetgrent(3N) |
| endnetpath(3N) | See getnetpath(3N) |
| endprotoent(3N) | See getprotobyname(3N) |
| endprotoent(3XN) | network protocol database functions |
| endpwent(3C) | See getpwnam(3C) |
| endrpcent(3N) | See getrpcbyname(3N) |
| endservent(3N) | See getservbyname(3N) |
| endservent(3XN) | network services database functions |
| endspent(3C) | See getspnam(3C) |
| endusershell(3C) | See getusershell(3C) |
| endutent(3C) | See getutent(3C) |
| endutxent(3C) | See getutxent(3C) |
| endwin(3X) | See curs_initscr(3X) |
| endwin(3XC) | restore initial terminal environment |
| erand48(3C) | See drand48(3C) |

| | |
|--------------------------|---|
| erase(3) | See plot(3) |
| erase(3X) | See curs_clear(3X) |
| erase(3XC) | See clear(3XC) |
| erasechar(3X) | See curs_termattrs(3X) |
| erasechar(3XC) | return current ERASE or KILL characters |
| eraseswchar(3XC) | See erasechar(3XC) |
| erf(3M) | error and complementary error functions |
| erfc(3M) | See erf(3M) |
| errno(3C) | See perror(3C) |
| etext(3C) | See end(3C) |
| ether_aton(3N) | See ethers(3N) |
| ether_hostton(3N) | See ethers(3N) |
| ether_line(3N) | See ethers(3N) |
| ether_ntoa(3N) | See ethers(3N) |
| ether_ntohost(3N) | See ethers(3N) |
| ethers(3N) | Ethernet address mapping operations |
| euccol(3C) | See euclen(3C) |
| euclen(3C) | get byte length and display width of EUC characters |
| eucscol(3C) | See euclen(3C) |
| exit(3C) | terminate process |
| exp(3M) | exponential function |
| expm1(3M) | computes exponential functions |

| | |
|-------------------------------|---|
| extended_to_decimal(3) | See floating_to_decimal(3) |
| fabs(3M) | absolute value function |
| fattach(3C) | attach a STREAMS-based file descriptor to an object in the file system name space |
| fclose(3S) | close a stream |
| fconvert(3) | See econvert(3) |
| fcvt(3) | See econvert(3) |
| fcvt(3C) | See ecvt(3C) |
| fdatasync(3R) | synchronize a file's data |
| fdetach(3C) | detach a name from a STREAMS-based file descriptor |
| fdopen(3S) | associate a stream with a file descriptor |
| feof(3S) | See ferror(3S) |
| ferror(3S) | stream status inquiries |
| fetch(3B) | See dbm(3B) |
| fflush(3S) | flush a stream |
| ffs(3C) | find first set bit |
| fgetc(3S) | get a byte from a stream |
| fgetgrent(3C) | See getgrnam(3C) |
| fgetgrent_r(3C) | See getgrnam(3C) |
| fgetpos(3S) | get current file position information |
| fgetpwent(3C) | See getpwnam(3C) |
| fgetpwent_r(3C) | See getpwnam(3C) |

| | |
|---------------------------|---|
| fgets(3S) | See gets(3S) |
| fgetspent(3C) | See getspnam(3C) |
| fgetspent_r(3C) | See getspnam(3C) |
| fgetwc(3S) | get a wide-character code from a stream |
| fgetws(3S) | See getws(3S) |
| field_arg(3X) | See form_field_validation(3X) |
| field_back(3X) | See form_field_attributes(3X) |
| field_buffer(3X) | See form_field_buffer(3X) |
| field_count(3X) | See form_field(3X) |
| field_fore(3X) | See form_field_attributes(3X) |
| field_index(3X) | See form_page(3X) |
| field_info(3X) | See form_field_info(3X) |
| field_init(3X) | See form_hook(3X) |
| field_just(3X) | See form_field_just(3X) |
| field_opts(3X) | See form_field_opts(3X) |
| field_opts_off(3X) | See form_field_opts(3X) |
| field_opts_on(3X) | See form_field_opts(3X) |
| field_pad(3X) | See form_field_attributes(3X) |
| field_status(3X) | See form_field_buffer(3X) |
| field_term(3X) | See form_hook(3X) |
| field_type(3X) | See form_field_validation(3X) |

| | |
|--------------------------------------|--|
| field_userptr(3X) | See form_field_userptr(3X) |
| file_to_decimal(3) | See string_to_decimal(3) |
| fileno(3S) | See ferror(3S) |
| filter(3X) | See curs_util(3X) |
| filter(3XC) | disable use of certain terminal capabilities |
| finite(3C) | See isnan(3C) |
| firstkey(3B) | See dbm(3B) |
| flash(3X) | See curs_beep(3X) |
| flash(3XC) | See beep(3XC) |
| floating_to_decimal(3) | convert floating-point value to decimal record |
| flock(3B) | apply or remove an advisory lock on an open file |
| flockfile(3S) | acquire and release stream lock |
| floor(3M) | floor function |
| flushinp(3X) | See curs_util(3X) |
| flushinp(3XC) | discard type-ahead characters |
| fmod(3M) | floating-point remainder value function |
| fmtmsg(3C) | display a message on stderr or system console |
| fn_attr_bind(3N) | bind a reference to a name and associate attributes with named object |
| fn_attr_create_subcontext(3N) | create a subcontext in a context and associate attributes with newly created context |

| | |
|--|---|
| <code>fn_attr_ext_search(3N)</code> | search for names in the specified context(s) whose attributes satisfy the filter |
| <code>fn_attr_get(3N)</code> | return specified attribute associated with name |
| <code>fn_attr_get_ids(3N)</code> | get a list of the identifiers of all attributes associated with named object |
| <code>fn_attr_get_values(3N)</code> | return values of an attribute |
| <code>fn_attr_modify(3N)</code> | modify specified attribute associated with name |
| <code>fn_attr_multi_get(3N)</code> | return multiple attributes associated with named object |
| <code>fn_attr_multi_modify(3N)</code> | modify multiple attributes associated with named object |
| <code>fn_attr_search(3N)</code> | search for the atomic name of objects with the specified attributes in a single context |
| <code>fn_attribute_add(3N)</code> | See <code>FN_attribute_t(3N)</code> |
| <code>fn_attribute_assign(3N)</code> | See <code>FN_attribute_t(3N)</code> |
| <code>fn_attribute_copy(3N)</code> | See <code>FN_attribute_t(3N)</code> |
| <code>fn_attribute_create(3N)</code> | See <code>FN_attribute_t(3N)</code> |
| <code>fn_attribute_destroy(3N)</code> | See <code>FN_attribute_t(3N)</code> |
| <code>fn_attribute_first(3N)</code> | See <code>FN_attribute_t(3N)</code> |
| <code>fn_attribute_identifier(3N)</code> | See <code>FN_attribute_t(3N)</code> |
| <code>fn_attribute_next(3N)</code> | See <code>FN_attribute_t(3N)</code> |
| <code>fn_attribute_remove(3N)</code> | See <code>FN_attribute_t(3N)</code> |
| <code>fn_attribute_syntax(3N)</code> | See <code>FN_attribute_t(3N)</code> |
| <code>fn_attribute_valuecount(3N)</code> | See <code>FN_attribute_t(3N)</code> |

| | |
|--|--|
| <code>fn_attrmodlist_add(3N)</code> | See <code>FN_attrmodlist_t(3N)</code> |
| <code>fn_attrmodlist_assign(3N)</code> | See <code>FN_attrmodlist_t(3N)</code> |
| <code>fn_attrmodlist_copy(3N)</code> | See <code>FN_attrmodlist_t(3N)</code> |
| <code>fn_attrmodlist_count(3N)</code> | See <code>FN_attrmodlist_t(3N)</code> |
| <code>fn_attrmodlist_create(3N)</code> | See <code>FN_attrmodlist_t(3N)</code> |
| <code>fn_attrmodlist_destroy(3N)</code> | See <code>FN_attrmodlist_t(3N)</code> |
| <code>fn_attrmodlist_first(3N)</code> | See <code>FN_attrmodlist_t(3N)</code> |
| <code>fn_attrmodlist_next(3N)</code> | See <code>FN_attrmodlist_t(3N)</code> |
| <code>fn_attrset_add(3N)</code> | See <code>FN_attrset_t(3N)</code> |
| <code>fn_attrset_assign(3N)</code> | See <code>FN_attrset_t(3N)</code> |
| <code>fn_attrset_copy(3N)</code> | See <code>FN_attrset_t(3N)</code> |
| <code>fn_attrset_count(3N)</code> | See <code>FN_attrset_t(3N)</code> |
| <code>fn_attrset_create(3N)</code> | See <code>FN_attrset_t(3N)</code> |
| <code>fn_attrset_destroy(3N)</code> | See <code>FN_attrset_t(3N)</code> |
| <code>fn_attrset_first(3N)</code> | See <code>FN_attrset_t(3N)</code> |
| <code>fn_attrset_get(3N)</code> | See <code>FN_attrset_t(3N)</code> |
| <code>fn_attrset_next(3N)</code> | See <code>FN_attrset_t(3N)</code> |
| <code>fn_attrset_remove(3N)</code> | See <code>FN_attrset_t(3N)</code> |
| <code>fn_bindinglist_destroy(3N)</code> | See <code>fn_ctx_list_bindings(3N)</code> |
| <code>fn_bindinglist_next(3N)</code> | See <code>fn_ctx_list_bindings(3N)</code> |
| <code>fn_composite_name_append_comp(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_append_name(3N)</code> | See <code>FN_composite_name_t(3N)</code> |

| | |
|--|---|
| <code>fn_composite_name_assign(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_copy(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_count(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_create(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_delete_comp(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_destroy(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_first(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_from_str(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_from_string(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_insert_comp(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_insert_name(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_is_empty(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_is_equal(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_is_prefix(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_is_suffix(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_last(3N)</code> | See <code>FN_composite_name_t(3N)</code> |

| | |
|---|---|
| <code>fn_composite_name_next(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_prefix(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_prepend_comp(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_prepend_name(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_prev(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_composite_name_suffix(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_compound_name_append_comp(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_compound_name_assign(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_compound_name_copy(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_compound_name_count(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_compound_name_delete_all(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_compound_name_delete_comp(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_compound_name_destroy(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_compound_name_first(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_compound_name_from_syntax_attrs(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_compound_name_get_syntax_attrs(3N)</code> | See <code>FN_compound_name_t(3N)</code> |

| | |
|--|--|
| <code>fn_compound_name_insert_comp(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_compound_name_is_empty(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_compound_name_is_equal(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_compound_name_is_prefix(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_compound_name_is_suffix(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_compound_name_last(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_compound_name_next(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_compound_name_prefix(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_compound_name_prepend_comp(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_compound_name_prev(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_compound_name_suffix(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_ctx_bind(3N)</code> | bind a reference to a name |
| <code>fn_ctx_create_subcontext(3N)</code> | create a subcontext in a context |
| <code>fn_ctx_destroy_subcontext(3N)</code> | destroy the named context and remove its binding from the parent context |
| <code>fn_ctx_equivalent_name(3N)</code> | construct an equivalent name in same context |
| <code>fn_ctx_get_ref(3N)</code> | return a context's reference |

| | |
|---|--|
| <code>fn_ctx_get_syntax_attrs(3N)</code> | return syntax attributes associated with named context |
| <code>fn_ctx_handle_destroy(3N)</code> | release storage associated with context handle |
| <code>fn_ctx_handle_from_initial(3N)</code> | return a handle to the Initial Context |
| <code>fn_ctx_handle_from_ref(3N)</code> | construct a handle to a context object using the given reference |
| <code>fn_ctx_list_bindings(3N)</code> | list the atomic names and references bound in a context |
| <code>fn_ctx_list_names(3N)</code> | list the atomic names bound in a context |
| <code>fn_ctx_lookup(3N)</code> | look up name in context |
| <code>fn_ctx_lookup_link(3N)</code> | look up the link reference bound to a name |
| <code>fn_ctx_rename(3N)</code> | rename the name of a binding |
| <code>fn_ctx_unbind(3N)</code> | unbind a name from a context |
| <code>fn_ext_searchlist_destroy(3N)</code> | See <code>fn_attr_ext_search(3N)</code> |
| <code>fn_ext_searchlist_next(3N)</code> | See <code>fn_attr_ext_search(3N)</code> |
| <code>fn_multigetlist_destroy(3N)</code> | See <code>fn_attr_multi_get(3N)</code> |
| <code>fn_multigetlist_next(3N)</code> | See <code>fn_attr_multi_get(3N)</code> |
| <code>fn_namelist_destroy(3N)</code> | See <code>fn_ctx_list_names(3N)</code> |
| <code>fn_namelist_next(3N)</code> | See <code>fn_ctx_list_names(3N)</code> |
| <code>fn_ref_addr_assign(3N)</code> | See <code>FN_ref_addr_t(3N)</code> |
| <code>fn_ref_addr_copy(3N)</code> | See <code>FN_ref_addr_t(3N)</code> |
| <code>fn_ref_addr_create(3N)</code> | See <code>FN_ref_addr_t(3N)</code> |
| <code>fn_ref_addr_data(3N)</code> | See <code>FN_ref_addr_t(3N)</code> |

| | |
|---|---|
| <code>fn_ref_addr_description(3N)</code> | See <code>FN_ref_addr_t(3N)</code> |
| <code>fn_ref_addr_destroy(3N)</code> | See <code>FN_ref_addr_t(3N)</code> |
| <code>fn_ref_addr_length(3N)</code> | See <code>FN_ref_addr_t(3N)</code> |
| <code>fn_ref_addr_type(3N)</code> | See <code>FN_ref_addr_t(3N)</code> |
| <code>fn_ref_addrcount(3N)</code> | See <code>FN_ref_t(3N)</code> |
| <code>fn_ref_append_addr(3N)</code> | See <code>FN_ref_t(3N)</code> |
| <code>fn_ref_assign(3N)</code> | See <code>FN_ref_t(3N)</code> |
| <code>fn_ref_copy(3N)</code> | See <code>FN_ref_t(3N)</code> |
| <code>fn_ref_create(3N)</code> | See <code>FN_ref_t(3N)</code> |
| <code>fn_ref_create_link(3N)</code> | See <code>FN_ref_t(3N)</code> |
| <code>fn_ref_delete_addr(3N)</code> | See <code>FN_ref_t(3N)</code> |
| <code>fn_ref_delete_all(3N)</code> | See <code>FN_ref_t(3N)</code> |
| <code>fn_ref_description(3N)</code> | See <code>FN_ref_t(3N)</code> |
| <code>fn_ref_destroy(3N)</code> | See <code>FN_ref_t(3N)</code> |
| <code>fn_ref_first(3N)</code> | See <code>FN_ref_t(3N)</code> |
| <code>fn_ref_insert_addr(3N)</code> | See <code>FN_ref_t(3N)</code> |
| <code>fn_ref_is_link(3N)</code> | See <code>FN_ref_t(3N)</code> |
| <code>fn_ref_link_name(3N)</code> | See <code>FN_ref_t(3N)</code> |
| <code>fn_ref_next(3N)</code> | See <code>FN_ref_t(3N)</code> |
| <code>fn_ref_prepend_addr(3N)</code> | See <code>FN_ref_t(3N)</code> |
| <code>fn_ref_type(3N)</code> | See <code>FN_ref_t(3N)</code> |
| <code>fn_search_control_assign(3N)</code> | See <code>FN_search_control_t(3N)</code> |
| <code>fn_search_control_copy(3N)</code> | See <code>FN_search_control_t(3N)</code> |

| | |
|--|---|
| <code>fn_search_control_create(3N)</code> | See <code>FN_search_control_t(3N)</code> |
| <code>fn_search_control_destroy(3N)</code> | See <code>FN_search_control_t(3N)</code> |
| <code>fn_search_control_follow_links(3N)</code> | See <code>FN_search_control_t(3N)</code> |
| <code>fn_search_control_max_names(3N)</code> | See <code>FN_search_control_t(3N)</code> |
| <code>fn_search_control_return_attr_ids(3N)</code> | See <code>FN_search_control_t(3N)</code> |
| <code>fn_search_control_return_ref(3N)</code> | See <code>FN_search_control_t(3N)</code> |
| <code>fn_search_control_scope(3N)</code> | See <code>FN_search_control_t(3N)</code> |
| <code>fn_search_filter_arguments(3N)</code> | See <code>FN_search_filter_t(3N)</code> |
| <code>fn_search_filter_assign(3N)</code> | See <code>FN_search_filter_t(3N)</code> |
| <code>fn_search_filter_copy(3N)</code> | See <code>FN_search_filter_t(3N)</code> |
| <code>fn_search_filter_create(3N)</code> | See <code>FN_search_filter_t(3N)</code> |
| <code>fn_search_filter_destroy(3N)</code> | See <code>FN_search_filter_t(3N)</code> |
| <code>fn_search_filter_expression(3N)</code> | See <code>FN_search_filter_t(3N)</code> |
| <code>fn_searchlist_destroy(3N)</code> | See <code>fn_attr_search(3N)</code> |
| <code>fn_searchlist_next(3N)</code> | See <code>fn_attr_search(3N)</code> |
| <code>fn_status_advance_by_name(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_append_remaining_name(3N)</code> | See <code>FN_status_t(3N)</code> |

| | |
|--|----------------------------------|
| <code>fn_status_append_resolved_name(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_assign(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_code(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_copy(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_create(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_description(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_destroy(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_diagnostic_message(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_is_success(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_link_code(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_link_diagnostic_message(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_link_remaining_name(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_link_resolved_name(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_link_resolved_ref(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_remaining_name(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_resolved_name(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_resolved_ref(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_set(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_set_code(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_set_diagnostic_message(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_set_link_code(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_set_link_diagnostic_message(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_set_link_remaining_name(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_set_link_resolved_name(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_set_link_resolved_ref(3N)</code> | See <code>FN_status_t(3N)</code> |

| | |
|--|---|
| <code>fn_status_set_remaining_name(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_set_resolved_name(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_set_resolved_ref(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_status_set_success(3N)</code> | See <code>FN_status_t(3N)</code> |
| <code>fn_string_assign(3N)</code> | See <code>FN_string_t(3N)</code> |
| <code>fn_string_bytecount(3N)</code> | See <code>FN_string_t(3N)</code> |
| <code>fn_string_charcount(3N)</code> | See <code>FN_string_t(3N)</code> |
| <code>fn_string_code_set(3N)</code> | See <code>FN_string_t(3N)</code> |
| <code>fn_string_compare(3N)</code> | See <code>FN_string_t(3N)</code> |
| <code>fn_string_compare_substring(3N)</code> | See <code>FN_string_t(3N)</code> |
| <code>fn_string_contents(3N)</code> | See <code>FN_string_t(3N)</code> |
| <code>fn_string_copy(3N)</code> | See <code>FN_string_t(3N)</code> |
| <code>fn_string_create(3N)</code> | See <code>FN_string_t(3N)</code> |
| <code>fn_string_destroy(3N)</code> | See <code>FN_string_t(3N)</code> |
| <code>fn_string_from_composite_name(3N)</code> | See <code>FN_composite_name_t(3N)</code> |
| <code>fn_string_from_compound_name(3N)</code> | See <code>FN_compound_name_t(3N)</code> |
| <code>fn_string_from_contents(3N)</code> | See <code>FN_string_t(3N)</code> |
| <code>fn_string_from_str(3N)</code> | See <code>FN_string_t(3N)</code> |
| <code>fn_string_from_str_n(3N)</code> | See <code>FN_string_t(3N)</code> |
| <code>fn_string_from_strings(3N)</code> | See <code>FN_string_t(3N)</code> |
| <code>fn_string_from_substring(3N)</code> | See <code>FN_string_t(3N)</code> |
| <code>fn_string_is_empty(3N)</code> | See <code>FN_string_t(3N)</code> |
| <code>fn_string_next_substring(3N)</code> | See <code>FN_string_t(3N)</code> |
| <code>fn_string_prev_substring(3N)</code> | See <code>FN_string_t(3N)</code> |

| | |
|--|---|
| <code>fn_string_str(3N)</code> | See <code>FN_string_t(3N)</code> |
| <code>fn_valuelist_destroy(3N)</code> | See <code>fn_attr_get_values(3N)</code> |
| <code>fn_valuelist_next(3N)</code> | See <code>fn_attr_get_values(3N)</code> |
| <code>fnmatch(3C)</code> | match filename or path name |
| <code>fopen(3B)</code> | open a stream |
| <code>fopen(3S)</code> | open a stream |
| <code>form_cursor(3X)</code> | position forms window cursor |
| <code>form_data(3X)</code> | tell if forms field has off-screen data ahead or behind |
| <code>form_driver(3X)</code> | command processor for the forms subsystem |
| <code>form_field(3X)</code> | connect fields to forms |
| <code>form_field_attributes(3X)</code> | format the general display attributes of forms |
| <code>form_field_buffer(3X)</code> | set and get forms field attributes |
| <code>form_field_info(3X)</code> | get forms field characteristics |
| <code>form_field_just(3X)</code> | format the general appearance of forms |
| <code>form_field_new(3X)</code> | create and destroy forms fields |
| <code>form_field_opts(3X)</code> | forms field option routines |
| <code>form_field_userptr(3X)</code> | associate application data with forms |
| <code>form_field_validation(3X)</code> | forms field data type validation |
| <code>form_fields(3X)</code> | See <code>form_field(3X)</code> |
| <code>form_fieldtype(3X)</code> | forms fieldtype routines |

| | |
|--------------------------------|--|
| <code>form_hook(3X)</code> | assign application-specific routines for invocation by forms |
| <code>form_init(3X)</code> | See <code>form_hook(3X)</code> |
| <code>form_new(3X)</code> | create and destroy forms |
| <code>form_new_page(3X)</code> | forms pagination |
| <code>form_opts(3X)</code> | forms option routines |
| <code>form_opts_off(3X)</code> | See <code>form_opts(3X)</code> |
| <code>form_opts_on(3X)</code> | See <code>form_opts(3X)</code> |
| <code>form_page(3X)</code> | set forms current page and field |
| <code>form_post(3X)</code> | write or erase forms from associated subwindows |
| <code>form_sub(3X)</code> | See <code>form_win(3X)</code> |
| <code>form_term(3X)</code> | See <code>form_hook(3X)</code> |
| <code>form_userptr(3X)</code> | associate application data with forms |
| <code>form_win(3X)</code> | forms window and subwindow association routines |
| <code>forms(3X)</code> | character based forms package |
| <code>fpclass(3C)</code> | See <code>isnan(3C)</code> |
| <code>fpgetmask(3C)</code> | See <code>fpgetround(3C)</code> |
| <code>fpgetround(3C)</code> | IEEE floating-point environment control |
| <code>fpgetsticky(3C)</code> | See <code>fpgetround(3C)</code> |
| <code>fprintf(3B)</code> | See <code>printf(3B)</code> |
| <code>fprintf(3S)</code> | See <code>printf(3S)</code> |
| <code>fpsetmask(3C)</code> | See <code>fpgetround(3C)</code> |
| <code>fpsetround(3C)</code> | See <code>fpgetround(3C)</code> |

| | |
|---------------------------|--|
| fpsetsticky(3C) | See fpgetround(3C) |
| fputc(3S) | put a byte on a stream |
| fputs(3S) | See puts(3S) |
| fputwc(3S) | put wide-character code on a stream |
| fputws(3S) | put wide character string on a stream |
| fread(3S) | buffered binary input/output |
| free(3C) | See malloc(3C) |
| free(3X) | See malloc(3X) |
| free(3t) | See mtmalloc(3t) |
| free_field(3X) | See form_field_new(3X) |
| free_fieldtype(3X) | See form_fieldtype(3X) |
| free_form(3X) | See form_new(3X) |
| free_item(3X) | See menu_item_new(3X) |
| free_menu(3X) | See menu_new(3X) |
| freenetconfigt(3N) | See getnetconfig(3N) |
| freopen(3B) | See fopen(3B) |
| freopen(3S) | open a stream |
| frexp(3C) | extract mantissa and exponent from double precision number |
| fscanf(3S) | See scanf(3S) |
| fseek(3S) | reposition a file-position indicator in a stream |
| fseeko(3S) | See fseek(3S) |
| fsetpos(3S) | reposition a file pointer in a stream |

| | |
|---------------------------|---|
| fsync(3C) | synchronize a file's in-memory state with that on the physical medium |
| ftell(3S) | return a file offset in a stream |
| ftello(3S) | See ftell(3S) |
| ftime(3C) | get date and time |
| ftok(3C) | generate an IPC key |
| ftruncate(3C) | See truncate(3C) |
| ftrylockfile(3S) | See flockfile(3S) |
| ftw(3C) | walk a file tree |
| func_to_decimal(3) | See string_to_decimal(3) |
| funlockfile(3S) | See flockfile(3S) |
| fwide(3C) | set stream orientation |
| fwprintf(3S) | print formatted wide-character output |
| fwrite(3S) | See fread(3S) |
| fwscanf(3S) | convert formatted wide-character input |
| gamma(3M) | See lgamma(3M) |
| gamma_r(3M) | See lgamma(3M) |
| gconvert(3) | See econvert(3) |
| gcvt(3) | See econvert(3) |
| gcvt(3C) | See ecvt(3C) |
| gelf(3E) | generic class-independent ELF interface |
| gelf_fsize(3E) | See gelf(3E) |
| gelf_getclass(3E) | See gelf(3E) |

| | |
|--------------------------------------|------------------------------------|
| <code>gelf_getdyn(3E)</code> | See <code>gelf(3E)</code> |
| <code>gelf_getehdr(3E)</code> | See <code>gelf(3E)</code> |
| <code>gelf_getphdr(3E)</code> | See <code>gelf(3E)</code> |
| <code>gelf_getrel(3E)</code> | See <code>gelf(3E)</code> |
| <code>gelf_getrela(3E)</code> | See <code>gelf(3E)</code> |
| <code>gelf_getshdr(3E)</code> | See <code>gelf(3E)</code> |
| <code>gelf_getsym(3E)</code> | See <code>gelf(3E)</code> |
| <code>gelf_getsyminfo(3E)</code> | See <code>gelf(3E)</code> |
| <code>gelf_newehdr(3E)</code> | See <code>gelf(3E)</code> |
| <code>gelf_newphdr(3E)</code> | See <code>gelf(3E)</code> |
| <code>gelf_update_dyn(3E)</code> | See <code>gelf(3E)</code> |
| <code>gelf_update_ehdr(3E)</code> | See <code>gelf(3E)</code> |
| <code>gelf_update_phdr(3E)</code> | See <code>gelf(3E)</code> |
| <code>gelf_update_rel(3E)</code> | See <code>gelf(3E)</code> |
| <code>gelf_update_rela(3E)</code> | See <code>gelf(3E)</code> |
| <code>gelf_update_shdr(3E)</code> | See <code>gelf(3E)</code> |
| <code>gelf_update_sym(3E)</code> | See <code>gelf(3E)</code> |
| <code>gelf_update_syminfo(3E)</code> | See <code>gelf(3E)</code> |
| <code>gelf_xlatetof(3E)</code> | See <code>gelf(3E)</code> |
| <code>gelf_xslatetom(3E)</code> | See <code>gelf(3E)</code> |
| <code>get_myaddress(3N)</code> | See <code>rpc_soc(3N)</code> |
| <code>get_wch(3XC)</code> | get a wide character from terminal |
| <code>get_wstr(3XC)</code> | See <code>getn_wstr(3XC)</code> |
| <code>getacdir(3)</code> | See <code>getacinfo(3)</code> |

| | |
|-----------------------------------|------------------------------------|
| <code>getacflg(3)</code> | See <code>getacinfo(3)</code> |
| <code>getacinfo(3)</code> | get audit control file information |
| <code>getacmin(3)</code> | See <code>getacinfo(3)</code> |
| <code>getacna(3)</code> | See <code>getacinfo(3)</code> |
| <code>getauclassent(3)</code> | get <code>audit_class</code> entry |
| <code>getauclassent_r(3)</code> | See <code>getauclassent(3)</code> |
| <code>getauclassnam(3)</code> | See <code>getauclassent(3)</code> |
| <code>getauclassnam_r(3)</code> | See <code>getauclassent(3)</code> |
| <code>getauditflags(3)</code> | convert audit flag specifications |
| <code>getauditflagsbin(3)</code> | See <code>getauditflags(3)</code> |
| <code>getauditflagschar(3)</code> | See <code>getauditflags(3)</code> |
| <code>getauevent(3)</code> | get <code>audit_event</code> entry |
| <code>getauevent_r(3)</code> | See <code>getauevent(3)</code> |
| <code>getauevnam(3)</code> | See <code>getauevent(3)</code> |
| <code>getauevnam_r(3)</code> | See <code>getauevent(3)</code> |
| <code>getauevnonam(3)</code> | See <code>getauevent(3)</code> |
| <code>getauevnum(3)</code> | See <code>getauevent(3)</code> |
| <code>getauevnum_r(3)</code> | See <code>getauevent(3)</code> |
| <code>getauuserent(3)</code> | See <code>getauusername(3)</code> |
| <code>getauusername(3)</code> | get <code>audit_user</code> entry |
| <code>getbegyx(3X)</code> | See <code>curs_getyx(3X)</code> |
| <code>getbegyx(3XC)</code> | get cursor or window coordinates |
| <code>getbkgrnd(3XC)</code> | See <code>bkgrnd(3XC)</code> |

| | |
|-----------------------------------|---|
| <code>getc(3S)</code> | See <code>fgetc(3S)</code> |
| <code>getc_unlocked(3S)</code> | See <code>fgetc(3S)</code> |
| <code>getcchar(3XC)</code> | get a wide character string (with rendition) from a <code>cchar_t</code> |
| <code>getch(3X)</code> | See <code>curs_getch(3X)</code> |
| <code>getch(3XC)</code> | get a single-byte character from terminal |
| <code>getchar(3S)</code> | See <code>fgetc(3S)</code> |
| <code>getchar_unlocked(3S)</code> | See <code>fgetc(3S)</code> |
| <code>getcwd(3C)</code> | get pathname of current working directory |
| <code>getdate(3C)</code> | convert user format date and time |
| <code>getdtablesize(3C)</code> | get the file descriptor table size |
| <code>getenv(3C)</code> | return value for environment name |
| <code>getexecname(3C)</code> | return pathname of executable |
| <code>getfauditflags(3)</code> | generates the process audit state |
| <code>getgrent(3C)</code> | See <code>getgrnam(3C)</code> |
| <code>getgrent_r(3C)</code> | See <code>getgrnam(3C)</code> |
| <code>getgrgid(3C)</code> | See <code>getgrnam(3C)</code> |
| <code>getgrgid_r(3C)</code> | See <code>getgrnam(3C)</code> |
| <code>getgrnam(3C)</code> | get group entry |
| <code>getgrnam_r(3C)</code> | See <code>getgrnam(3C)</code> |
| <code>gethostbyaddr(3N)</code> | See <code>gethostbyname(3N)</code> |
| <code>gethostbyaddr(3XN)</code> | See <code>endhostent(3XN)</code> |
| <code>gethostbyaddr_r(3N)</code> | See <code>gethostbyname(3N)</code> |

| | |
|----------------------------------|---|
| <code>gethostbyname(3N)</code> | get network host entry |
| <code>gethostbyname(3XN)</code> | See <code>endhostent(3XN)</code> |
| <code>gethostbyname_r(3N)</code> | See <code>gethostbyname(3N)</code> |
| <code>gethostent(3N)</code> | See <code>gethostbyname(3N)</code> |
| <code>gethostent(3XN)</code> | See <code>endhostent(3XN)</code> |
| <code>gethostent_r(3N)</code> | See <code>gethostbyname(3N)</code> |
| <code>gethostid(3C)</code> | get unique identifier of current host |
| <code>gethostname(3C)</code> | get or set name of current host |
| <code>gethostname(3XN)</code> | get name of current host |
| <code>gethrtime(3C)</code> | get high resolution time |
| <code>gethrvtime(3C)</code> | See <code>gethrtime(3C)</code> |
| <code>getloadavg(3C)</code> | get system load averages |
| <code>getlogin(3C)</code> | get login name |
| <code>getlogin_r(3C)</code> | See <code>getlogin(3C)</code> |
| <code>getmaxyx(3X)</code> | See <code>curs_getyx(3X)</code> |
| <code>getmaxyx(3XC)</code> | See <code>getbegyx(3XC)</code> |
| <code>getmntany(3C)</code> | See <code>getmntent(3C)</code> |
| <code>getmntent(3C)</code> | get mnttab file information |
| <code>getn_wstr(3XC)</code> | get a wide character string from terminal |
| <code>getnetbyaddr(3N)</code> | See <code>getnetbyname(3N)</code> |
| <code>getnetbyaddr(3XN)</code> | See <code>endnetent(3XN)</code> |
| <code>getnetbyaddr_r(3N)</code> | See <code>getnetbyname(3N)</code> |
| <code>getnetbyname(3N)</code> | get network entry |

| | |
|----------------------------------|---|
| <code>getnetbyname(3XN)</code> | See <code>endnetent(3XN)</code> |
| <code>getnetbyname_r(3N)</code> | See <code>getnetbyname(3N)</code> |
| <code>getnetconfig(3N)</code> | get network configuration database entry |
| <code>getnetconfigent(3N)</code> | See <code>getnetconfig(3N)</code> |
| <code>getnetent(3N)</code> | See <code>getnetbyname(3N)</code> |
| <code>getnetent(3XN)</code> | See <code>endnetent(3XN)</code> |
| <code>getnetent_r(3N)</code> | See <code>getnetbyname(3N)</code> |
| <code>getnetgrent(3N)</code> | get network group entry |
| <code>getnetgrent_r(3N)</code> | See <code>getnetgrent(3N)</code> |
| <code>getnetname(3N)</code> | See <code>secure_rpc(3N)</code> |
| <code>getnetpath(3N)</code> | get /etc/netconfig entry corresponding to NETPATH component |
| <code>getnstr(3XC)</code> | get a multibyte character string from terminal |
| <code>getnwstr(3X)</code> | See <code>curs_getwstr(3X)</code> |
| <code>getopt(3C)</code> | get option letter from argument vector |
| <code>getpagesize(3C)</code> | get system page size |
| <code>getparyx(3X)</code> | See <code>curs_getyx(3X)</code> |
| <code>getparyx(3XC)</code> | See <code>getbegyx(3XC)</code> |
| <code>getpass(3C)</code> | read a string of characters without echo |
| <code>getpassphrase(3C)</code> | See <code>getpass(3C)</code> |
| <code>getpeername(3N)</code> | get name of connected peer |
| <code>getpeername(3XN)</code> | get the name of the peer socket |

| | |
|-------------------------------------|--|
| <code>getpriority(3C)</code> | get or set process scheduling priority |
| <code>getprotobyname(3N)</code> | get protocol entry |
| <code>getprotobyname(3XN)</code> | See <code>endprotoent(3XN)</code> |
| <code>getprotobyname_r(3N)</code> | See <code>getprotobyname(3N)</code> |
| <code>getprotobynumber(3N)</code> | See <code>getprotobyname(3N)</code> |
| <code>getprotobynumber(3XN)</code> | See <code>endprotoent(3XN)</code> |
| <code>getprotobynumber_r(3N)</code> | See <code>getprotobyname(3N)</code> |
| <code>getprotoent(3N)</code> | See <code>getprotobyname(3N)</code> |
| <code>getprotoent(3XN)</code> | See <code>endprotoent(3XN)</code> |
| <code>getprotoent_r(3N)</code> | See <code>getprotobyname(3N)</code> |
| <code>getpublickey(3N)</code> | retrieve public or secret key |
| <code>getpw(3C)</code> | get passwd entry from UID |
| <code>getpwent(3C)</code> | See <code>getpwnam(3C)</code> |
| <code>getpwent_r(3C)</code> | See <code>getpwnam(3C)</code> |
| <code>getpwnam(3C)</code> | get password entry |
| <code>getpwnam_r(3C)</code> | See <code>getpwnam(3C)</code> |
| <code>getpwuid(3C)</code> | See <code>getpwnam(3C)</code> |
| <code>getpwuid_r(3C)</code> | See <code>getpwnam(3C)</code> |
| <code>getrpcbyname(3N)</code> | get RPC entry |
| <code>getrpcbyname_r(3N)</code> | See <code>getrpcbyname(3N)</code> |
| <code>getrpcbynumber(3N)</code> | See <code>getrpcbyname(3N)</code> |
| <code>getrpcbynumber_r(3N)</code> | See <code>getrpcbyname(3N)</code> |
| <code>getrpcent(3N)</code> | See <code>getrpcbyname(3N)</code> |
| <code>getrpcent_r(3N)</code> | See <code>getrpcbyname(3N)</code> |

| | |
|----------------------------------|--|
| <code>getrpcport(3N)</code> | See <code>rpc_soc(3N)</code> |
| <code>getrusage(3C)</code> | get information about resource utilization |
| <code>gets(3S)</code> | get a string from a stream |
| <code>getsecretkey(3N)</code> | See <code>getpublickey(3N)</code> |
| <code>getservbyname(3N)</code> | get service entry |
| <code>getservbyname(3XN)</code> | See <code>endservent(3XN)</code> |
| <code>getservbyname_r(3N)</code> | See <code>getservbyname(3N)</code> |
| <code>getservbyport(3N)</code> | See <code>getservbyname(3N)</code> |
| <code>getservbyport(3XN)</code> | See <code>endservent(3XN)</code> |
| <code>getservbyport_r(3N)</code> | See <code>getservbyname(3N)</code> |
| <code>getservent(3N)</code> | See <code>getservbyname(3N)</code> |
| <code>getservent(3XN)</code> | See <code>endservent(3XN)</code> |
| <code>getservent_r(3N)</code> | See <code>getservbyname(3N)</code> |
| <code>getsockname(3N)</code> | get socket name |
| <code>getsockname(3XN)</code> | get the socket name |
| <code>getsockopt(3N)</code> | get and set options on sockets |
| <code>getsockopt(3XN)</code> | get the socket options |
| <code>getspent(3C)</code> | See <code>getspnam(3C)</code> |
| <code>getspent_r(3C)</code> | See <code>getspnam(3C)</code> |
| <code>getspnam(3C)</code> | get password entry |
| <code>getspnam_r(3C)</code> | See <code>getspnam(3C)</code> |
| <code>getstr(3X)</code> | See <code>curs_getstr(3X)</code> |
| <code>getstr(3XC)</code> | See <code>getnstr(3XC)</code> |
| <code>getsubopt(3C)</code> | parse suboptions from a string |

| | |
|-------------------------|--|
| getsyx(3X) | See curs_kernel(3X) |
| gettext(3C) | message handling functions |
| gettimeofday(3B) | get or set the date and time |
| gettimeofday(3C) | get or set the date and time |
| gettxt(3C) | retrieve a text string |
| getusershell(3C) | get legal user shells |
| getutent(3C) | access utmp file entry |
| getutid(3C) | See getutent(3C) |
| getutline(3C) | See getutent(3C) |
| getutmp(3C) | See getutxent(3C) |
| getutmpx(3C) | See getutxent(3C) |
| getutxent(3C) | access utmpx file entry |
| getutxid(3C) | See getutxent(3C) |
| getutxline(3C) | See getutxent(3C) |
| getvfsany(3C) | See getvfSENT(3C) |
| getvfSENT(3C) | get vfstab file entry |
| getvfSfile(3C) | See getvfSENT(3C) |
| getvfSspec(3C) | See getvfSENT(3C) |
| getw(3S) | See fgetc(3S) |
| getwc(3S) | get wide character from a stream |
| getwch(3X) | See curs_getwch(3X) |
| getwchar(3S) | get wide character from stdin stream |
| getwd(3C) | get current working directory pathname |

| | |
|------------------------------|--|
| getwidth(3C) | get codeset information |
| getwin(3X) | See curs_util(3X) |
| getwin(3XC) | read a window from, and write a window to, a file |
| getws(3S) | convert a string of EUC characters from the stream to Process Code |
| getwstr(3X) | See curs_getwstr(3X) |
| getyx(3X) | See curs_getyx(3X) |
| getyx(3XC) | See getbegyx(3XC) |
| glob(3C) | generate path names matching a pattern |
| global_variables(3XC) | variables used for X/Open Curses |
| globfree(3C) | See glob(3C) |
| gmatch(3G) | shell global pattern matching |
| gmtime(3C) | See ctime(3C) |
| gmtime_r(3C) | See ctime(3C) |
| grantpt(3C) | grant access to the slave pseudo-terminal device |
| gsignal(3C) | See ssignal(3C) |
| halfdelay(3X) | See curs_inopts(3X) |
| halfdelay(3XC) | enable/disable half-delay mode |
| has_colors(3X) | See curs_color(3X) |
| has_colors(3XC) | See can_change_color(3XC) |
| has_ic(3X) | See curs_termattrs(3X) |
| has_ic(3XC) | determine insert/delete character/line capability |

| | |
|-------------------------|--|
| has_il(3X) | See curs_termattrs(3X) |
| has_il(3XC) | See has_ic(3XC) |
| hasmntopt(3C) | See getmntent(3C) |
| havedisk(3N) | See rstat(3N) |
| hcreate(3C) | See hsearch(3C) |
| hdestroy(3C) | See hsearch(3C) |
| hide_panel(3X) | See panel_show(3X) |
| hline(3XC) | use single-byte characters (and renditions) to draw lines |
| hline_set(3XC) | use complex characters (and renditions) to draw lines |
| host2netname(3N) | See secure_rpc(3N) |
| hsearch(3C) | manage hash search tables |
| htonl(3N) | See byteorder(3N) |
| htonl(3XN) | convert values between host and network byte order |
| htons(3N) | See byteorder(3N) |
| htons(3XN) | See htonl(3XN) |
| hypot(3M) | Euclidean distance function |
| iconv(3) | code conversion function |
| iconv_close(3) | code conversion deallocation function |
| iconv_open(3) | code conversion allocation function |
| idcok(3X) | See curs_ouptops(3X) |
| idcok(3XC) | enable/disable hardware insert-character and delete-character features |

| | |
|---------------------------------|--|
| <code>idlok(3X)</code> | See <code>curls_outopts(3X)</code> |
| <code>idlok(3XC)</code> | See <code>clearok(3XC)</code> |
| <code>ilogb(3M)</code> | returns an unbiased exponent |
| <code>immedok(3X)</code> | See <code>curls_outopts(3X)</code> |
| <code>immedok(3XC)</code> | call refresh on changes to window |
| <code>in_wch(3XC)</code> | retrieve a complex character (with rendition) |
| <code>in_wchnstr(3XC)</code> | retrieve complex character string (with rendition) |
| <code>in_wchstr(3XC)</code> | See <code>in_wchnstr(3XC)</code> |
| <code>inch(3X)</code> | See <code>curls_inch(3X)</code> |
| <code>inch(3XC)</code> | return a single-byte character (with rendition) |
| <code>inchnstr(3X)</code> | See <code>curls_inchstr(3X)</code> |
| <code>inchnstr(3XC)</code> | retrieve a single-byte character string (with rendition) |
| <code>inchstr(3X)</code> | See <code>curls_inchstr(3X)</code> |
| <code>inchstr(3XC)</code> | See <code>inchnstr(3XC)</code> |
| <code>index(3C)</code> | string operations |
| <code>inet(3N)</code> | Internet address manipulation |
| <code>inet_addr(3N)</code> | See <code>inet(3N)</code> |
| <code>inet_addr(3XN)</code> | Internet address manipulation |
| <code>inet_lnaof(3N)</code> | See <code>inet(3N)</code> |
| <code>inet_lnaof(3XN)</code> | See <code>inet_addr(3XN)</code> |
| <code>inet_makeaddr(3N)</code> | See <code>inet(3N)</code> |
| <code>inet_makeaddr(3XN)</code> | See <code>inet_addr(3XN)</code> |

| | |
|--------------------------------|--|
| <code>inet_netof(3N)</code> | See <code>inet(3N)</code> |
| <code>inet_netof(3XN)</code> | See <code>inet_addr(3XN)</code> |
| <code>inet_network(3N)</code> | See <code>inet(3N)</code> |
| <code>inet_network(3XN)</code> | See <code>inet_addr(3XN)</code> |
| <code>inet_ntoa(3N)</code> | See <code>inet(3N)</code> |
| <code>inet_ntoa(3XN)</code> | See <code>inet_addr(3XN)</code> |
| <code>init_color(3X)</code> | See <code>curl_color(3X)</code> |
| <code>init_color(3XC)</code> | See <code>can_change_color(3XC)</code> |
| <code>init_pair(3X)</code> | See <code>curl_color(3X)</code> |
| <code>init_pair(3XC)</code> | See <code>can_change_color(3XC)</code> |
| <code>initgroups(3C)</code> | initialize the supplementary group access list |
| <code>initscr(3X)</code> | See <code>curl_initscr(3X)</code> |
| <code>initscr(3XC)</code> | screen initialization functions |
| <code>initstate(3C)</code> | See <code>random(3C)</code> |
| <code>innetgr(3N)</code> | See <code>getnetgrent(3N)</code> |
| <code>innstr(3X)</code> | See <code>curl_instr(3X)</code> |
| <code>innstr(3XC)</code> | retrieve a multibyte character string (without rendition) |
| <code>innwstr(3X)</code> | See <code>curl_inwstr(3X)</code> |
| <code>innwstr(3XC)</code> | retrieve a wide character string (without rendition) |
| <code>ins_nwstr(3XC)</code> | insert a wide character string |
| <code>ins_wch(3XC)</code> | insert a complex character |
| <code>ins_wstr(3XC)</code> | See <code>ins_nwstr(3XC)</code> |
| <code>insch(3X)</code> | See <code>curl_insch(3X)</code> |

| | |
|-----------------------------|--|
| <code>insch(3XC)</code> | insert a character |
| <code>insdelln(3X)</code> | See <code>curls_deleteln(3X)</code> |
| <code>insdelln(3XC)</code> | insert/delete lines to/from the window |
| <code>insertln(3X)</code> | See <code>curls_deleteln(3X)</code> |
| <code>insertln(3XC)</code> | insert a line in a window |
| <code>insnstr(3X)</code> | See <code>curls_insnstr(3X)</code> |
| <code>insnstr(3XC)</code> | insert a multibyte character string |
| <code>insnwstr(3X)</code> | See <code>curls_innwstr(3X)</code> |
| <code>insque(3C)</code> | insert/remove element from a queue |
| <code>insstr(3X)</code> | See <code>curls_insnstr(3X)</code> |
| <code>insstr(3XC)</code> | See <code>insnstr(3XC)</code> |
| <code>instr(3X)</code> | See <code>curls_instr(3X)</code> |
| <code>instr(3XC)</code> | See <code>innstr(3XC)</code> |
| <code>inwch(3X)</code> | See <code>curls_inwch(3X)</code> |
| <code>inwstr(3X)</code> | See <code>curls_innwstr(3X)</code> |
| <code>intrflush(3X)</code> | See <code>curls_inopts(3X)</code> |
| <code>intrflush(3XC)</code> | flush output in tty on interrupt |
| <code>intro(3)</code> | See <code>Intro(3)</code> |
| <code>inwch(3X)</code> | See <code>curls_inwch(3X)</code> |
| <code>inwchnstr(3X)</code> | See <code>curls_inwchstr(3X)</code> |
| <code>inwchstr(3X)</code> | See <code>curls_inwchstr(3X)</code> |
| <code>inwstr(3X)</code> | See <code>curls_innwstr(3X)</code> |
| <code>inwstr(3XC)</code> | See <code>innwstr(3XC)</code> |

| | |
|----------------------------------|---|
| <code>is_linetouched(3X)</code> | See <code>curs_touch(3X)</code> |
| <code>is_linetouched(3XC)</code> | control window refresh |
| <code>is_wintouched(3X)</code> | See <code>curs_touch(3X)</code> |
| <code>is_wintouched(3XC)</code> | See <code>is_linetouched(3XC)</code> |
| <code>isaexec(3C)</code> | invoke isa-specific executable |
| <code>isalnum(3C)</code> | See <code>ctype(3C)</code> |
| <code>isalpha(3C)</code> | See <code>ctype(3C)</code> |
| <code>isascii(3C)</code> | See <code>ctype(3C)</code> |
| <code>isastream(3C)</code> | test a file descriptor |
| <code>isatty(3C)</code> | test for a terminal device |
| <code>iscntrl(3C)</code> | See <code>ctype(3C)</code> |
| <code>isdigit(3C)</code> | See <code>ctype(3C)</code> |
| <code>isencrypt(3G)</code> | determine whether a buffer of characters is encrypted |
| <code>isendwin(3X)</code> | See <code>curs_initscr(3X)</code> |
| <code>isendwin(3XC)</code> | See <code>endwin(3XC)</code> |
| <code>isenglish(3C)</code> | See <code>iswalpha(3C)</code> |
| <code>isgraph(3C)</code> | See <code>ctype(3C)</code> |
| <code>isideogram(3C)</code> | See <code>iswalpha(3C)</code> |
| <code>islower(3C)</code> | See <code>ctype(3C)</code> |
| <code>isnan(3C)</code> | determine type of floating-point number |
| <code>isnan(3M)</code> | test for NaN |
| <code>isnand(3C)</code> | See <code>isnan(3C)</code> |
| <code>isnanf(3C)</code> | See <code>isnan(3C)</code> |
| <code>isnumber(3C)</code> | See <code>iswalpha(3C)</code> |

| | |
|-----------------------------------|---|
| <code>isphonogram(3C)</code> | See <code>iswalpha(3C)</code> |
| <code>isprint(3C)</code> | See <code>ctype(3C)</code> |
| <code>ispunct(3C)</code> | See <code>ctype(3C)</code> |
| <code>isspace(3C)</code> | See <code>ctype(3C)</code> |
| <code>isspecial(3C)</code> | See <code>iswalpha(3C)</code> |
| <code>isupper(3C)</code> | See <code>ctype(3C)</code> |
| <code>iswalnum(3C)</code> | See <code>iswalpha(3C)</code> |
| <code>iswalpha(3C)</code> | wide-character code classification functions |
| <code>iswascii(3C)</code> | See <code>iswalpha(3C)</code> |
| <code>iswcntrl(3C)</code> | See <code>iswalpha(3C)</code> |
| <code>iswctype(3C)</code> | test character for specified class |
| <code>iswdigit(3C)</code> | See <code>iswalpha(3C)</code> |
| <code>iswgraph(3C)</code> | See <code>iswalpha(3C)</code> |
| <code>iswlower(3C)</code> | See <code>iswalpha(3C)</code> |
| <code>iswprint(3C)</code> | See <code>iswalpha(3C)</code> |
| <code>iswpunct(3C)</code> | See <code>iswalpha(3C)</code> |
| <code>iswspace(3C)</code> | See <code>iswalpha(3C)</code> |
| <code>iswupper(3C)</code> | See <code>iswalpha(3C)</code> |
| <code>iswxdigit(3C)</code> | See <code>iswalpha(3C)</code> |
| <code>isxdigit(3C)</code> | See <code>ctype(3C)</code> |
| <code>item_count(3X)</code> | See <code>menu_items(3X)</code> |
| <code>item_description(3X)</code> | See <code>menu_item_name(3X)</code> |
| <code>item_index(3X)</code> | See <code>menu_item_current(3X)</code> |
| <code>item_init(3X)</code> | See <code>menu_hook(3X)</code> |

| | |
|---------------------------------------|---|
| <code>item_name(3X)</code> | See <code>menu_item_name(3X)</code> |
| <code>item_opts(3X)</code> | See <code>menu_item_opts(3X)</code> |
| <code>item_opts_off(3X)</code> | See <code>menu_item_opts(3X)</code> |
| <code>item_opts_on(3X)</code> | See <code>menu_item_opts(3X)</code> |
| <code>item_term(3X)</code> | See <code>menu_hook(3X)</code> |
| <code>item_userptr(3X)</code> | See <code>menu_item_userptr(3X)</code> |
| <code>item_value(3X)</code> | See <code>menu_item_value(3X)</code> |
| <code>item_visible(3X)</code> | See <code>menu_item_visible(3X)</code> |
| <code>j0(3M)</code> | Bessel functions of the first kind |
| <code>j1(3M)</code> | See <code>j0(3M)</code> |
| <code>jn(3M)</code> | See <code>j0(3M)</code> |
| <code>jrand48(3C)</code> | See <code>drand48(3C)</code> |
| <code>kerberos(3N)</code> | Kerberos authentication library |
| <code>kerberos_rpc(3N)</code> | library routines for remote procedure calls using Kerberos authentication |
| <code>key_decryptsession(3N)</code> | See <code>secure_rpc(3N)</code> |
| <code>key_encryptsession(3N)</code> | See <code>secure_rpc(3N)</code> |
| <code>key_gendes(3N)</code> | See <code>secure_rpc(3N)</code> |
| <code>key_name(3XC)</code> | See <code>keyname(3XC)</code> |
| <code>key_secretkey_is_set(3N)</code> | See <code>secure_rpc(3N)</code> |
| <code>key_setsecret(3N)</code> | See <code>secure_rpc(3N)</code> |
| <code>keyname(3X)</code> | See <code>curs_util(3X)</code> |
| <code>keyname(3XC)</code> | return character string used as key name |
| <code>keypad(3X)</code> | See <code>curs_inopts(3X)</code> |

| | |
|----------------------------|---|
| keypad(3XC) | enable/disable keypad handling |
| killchar(3X) | See curs_termattrs(3X) |
| killchar(3XC) | See erasechar(3XC) |
| killpg(3C) | send signal to a process group |
| killwchar(3XC) | See erasechar(3XC) |
| krb_get_admhst(3N) | See krb_realmofhost(3N) |
| krb_get_cred(3N) | See kerberos(3N) |
| krb_get_krbhst(3N) | See krb_realmofhost(3N) |
| krb_get_lrealm(3N) | See krb_realmofhost(3N) |
| krb_get_phost(3N) | See krb_realmofhost(3N) |
| krb_kntoln(3N) | See kerberos(3N) |
| krb_mk_err(3N) | See kerberos(3N) |
| krb_mk_req(3N) | See kerberos(3N) |
| krb_mk_safe(3N) | See kerberos(3N) |
| krb_net_read(3N) | See krb_sendauth(3N) |
| krb_net_write(3N) | See krb_sendauth(3N) |
| krb_rd_err(3N) | See kerberos(3N) |
| krb_rd_req(3N) | See kerberos(3N) |
| krb_rd_safe(3N) | See kerberos(3N) |
| krb_realmofhost(3N) | additional Kerberos utility routines |
| krb_recvauth(3N) | See krb_sendauth(3N) |
| krb_sendauth(3N) | Kerberos routines for sending authentication via network stream sockets |
| krb_set_key(3N) | See kerberos(3N) |

| | |
|-------------------------------------|---|
| <code>krb_set_tkt_string(3N)</code> | set Kerberos ticket cache file name |
| <code>kstat(3K)</code> | kernel statistics facility |
| <code>kstat_chain_update(3K)</code> | update the kstat header chain |
| <code>kstat_close(3K)</code> | See <code>kstat_open(3K)</code> |
| <code>kstat_data_lookup(3K)</code> | See <code>kstat_lookup(3K)</code> |
| <code>kstat_lookup(3K)</code> | find a kstat by name |
| <code>kstat_open(3K)</code> | initialize kernel statistics facility |
| <code>kstat_read(3K)</code> | read or write kstat data |
| <code>kstat_write(3K)</code> | See <code>kstat_read(3K)</code> |
| <code>kvm_close(3K)</code> | See <code>kvm_open(3K)</code> |
| <code>kvm_getcmd(3K)</code> | See <code>kvm_getu(3K)</code> |
| <code>kvm_getproc(3K)</code> | See <code>kvm_nextproc(3K)</code> |
| <code>kvm_getu(3K)</code> | get the u-area or invocation arguments for a process |
| <code>kvm_kread(3K)</code> | See <code>kvm_read(3K)</code> |
| <code>kvm_kwrite(3K)</code> | See <code>kvm_read(3K)</code> |
| <code>kvm_nextproc(3K)</code> | read system process structures |
| <code>kvm_nlist(3K)</code> | get entries from kernel symbol table |
| <code>kvm_open(3K)</code> | specify a kernel to examine |
| <code>kvm_read(3K)</code> | copy data to or from a kernel image or running system |
| <code>kvm_setproc(3K)</code> | See <code>kvm_nextproc(3K)</code> |
| <code>kvm_uread(3K)</code> | See <code>kvm_read(3K)</code> |
| <code>kvm_uwrite(3K)</code> | See <code>kvm_read(3K)</code> |

| | |
|-------------------------------------|---|
| <code>kvm_write(3K)</code> | See <code>kvm_read(3K)</code> |
| <code>l64a(3C)</code> | See <code>a641(3C)</code> |
| <code>label(3)</code> | See <code>plot(3)</code> |
| <code>labs(3C)</code> | See <code>abs(3C)</code> |
| <code>lckpwn(3C)</code> | manipulate shadow password database lock file |
| <code>lcong48(3C)</code> | See <code>drand48(3C)</code> |
| <code>ldap(3N)</code> | Lightweight Directory Access Protocol package |
| <code>ldap_8859_to_t61(3N)</code> | See <code>ldap_charset(3N)</code> |
| <code>ldap_abandon(3N)</code> | abandon an LDAP operation in progress |
| <code>ldap_add(3N)</code> | perform an LDAP add operation |
| <code>ldap_add_ext(3N)</code> | See <code>ldap_add(3N)</code> |
| <code>ldap_add_ext_s(3N)</code> | See <code>ldap_add(3N)</code> |
| <code>ldap_add_s(3N)</code> | See <code>ldap_add(3N)</code> |
| <code>ldap_bind(3N)</code> | LDAP bind functions |
| <code>ldap_bind_s(3N)</code> | See <code>ldap_bind(3N)</code> |
| <code>ldap_build_filter(3N)</code> | See <code>ldap_getfilter(3N)</code> |
| <code>ldap_cache(3N)</code> | LDAP client caching functions |
| <code>ldap_charset(3N)</code> | LDAP character set translation functions |
| <code>ldap_compare(3N)</code> | LDAP compare operation |
| <code>ldap_compare_ext(3N)</code> | See <code>ldap_compare(3N)</code> |
| <code>ldap_compare_ext_s(3N)</code> | See <code>ldap_compare(3N)</code> |
| <code>ldap_compare_s(3N)</code> | See <code>ldap_compare(3N)</code> |

| | |
|--|--|
| <code>ldap_control_free(3N)</code> | LDAP control disposal |
| <code>ldap_controls_free(3N)</code> | See <code>ldap_control_free(3N)</code> |
| <code>ldap_count_entries(3N)</code> | See <code>ldap_first_entry(3N)</code> |
| <code>ldap_count_message(3N)</code> | See <code>ldap_first_message(3N)</code> |
| <code>ldap_count_references(3N)</code> | See <code>ldap_first_entry(3N)</code> |
| <code>ldap_count_values(3N)</code> | See <code>ldap_get_values(3N)</code> |
| <code>ldap_count_values_len(3N)</code> | See <code>ldap_get_values(3N)</code> |
| <code>ldap_delete(3N)</code> | LDAP delete operation |
| <code>ldap_delete_ext(3N)</code> | See <code>ldap_delete(3N)</code> |
| <code>ldap_delete_ext_s(3N)</code> | See <code>ldap_delete(3N)</code> |
| <code>ldap_delete_s(3N)</code> | See <code>ldap_delete(3N)</code> |
| <code>ldap_destroy_cache(3N)</code> | See <code>ldap_cache(3N)</code> |
| <code>ldap_disable_cache(3N)</code> | See <code>ldap_cache(3N)</code> |
| <code>ldap_disptmpl(3N)</code> | LDAP display template functions |
| <code>ldap_dn2ufn(3N)</code> | See <code>ldap_get_dn(3N)</code> |
| <code>ldap_dn_to_url(3N)</code> | See <code>ldap_url(3N)</code> |
| <code>ldap_dns_to_dn(3N)</code> | See <code>ldap_get_dn(3N)</code> |
| <code>ldap_dns_to_url(3N)</code> | See <code>ldap_url(3N)</code> |
| <code>ldap_enable_cache(3N)</code> | See <code>ldap_cache(3N)</code> |
| <code>ldap_enable_translation(3N)</code> | See <code>ldap_charset(3N)</code> |
| <code>ldap_entry2html(3N)</code> | See <code>ldap_entry2text(3N)</code> |
| <code>ldap_entry2html_search(3N)</code> | See <code>ldap_entry2text(3N)</code> |
| <code>ldap_entry2text(3N)</code> | LDAP entry display functions |
| <code>ldap_entry2text_search(3N)</code> | See <code>ldap_entry2text(3N)</code> |

| | |
|--|---|
| <code>ldap_err2string(3N)</code> | See <code>ldap_error(3N)</code> |
| <code>ldap_errlist(3N)</code> | See <code>ldap_error(3N)</code> |
| <code>ldap_error(3N)</code> | LDAP protocol error handling functions |
| <code>ldap_explode_dn(3N)</code> | See <code>ldap_get_dn(3N)</code> |
| <code>ldap_explode_dns(3N)</code> | See <code>ldap_get_dn(3N)</code> |
| <code>ldap_first_attribute(3N)</code> | step through LDAP entry attributes |
| <code>ldap_first_disptmpl(3N)</code> | See <code>ldap_disptmpl(3N)</code> |
| <code>ldap_first_entry(3N)</code> | LDAP entry parsing and counting functions |
| <code>ldap_first_message(3N)</code> | LDAP message processing functions |
| <code>ldap_first_reference(3N)</code> | See <code>ldap_first_entry(3N)</code> |
| <code>ldap_first_reference(3N)</code> | See <code>ldap_first_entry(3N)</code> |
| <code>ldap_first_searchobj(3N)</code> | See <code>ldap_searchprefs(3N)</code> |
| <code>ldap_first_tmplcol(3N)</code> | See <code>ldap_disptmpl(3N)</code> |
| <code>ldap_first_tmplrow(3N)</code> | See <code>ldap_disptmpl(3N)</code> |
| <code>ldap_flush_cache(3N)</code> | See <code>ldap_cache(3N)</code> |
| <code>ldap_free_friendlymap(3N)</code> | See <code>ldap_friendly(3N)</code> |
| <code>ldap_free_searchprefs(3N)</code> | See <code>ldap_searchprefs(3N)</code> |
| <code>ldap_free_templates(3N)</code> | See <code>ldap_disptmpl(3N)</code> |
| <code>ldap_free_urldesc(3N)</code> | See <code>ldap_url(3N)</code> |
| <code>ldap_friendly(3N)</code> | LDAP attribute remapping functions |
| <code>ldap_friendly_name(3N)</code> | See <code>ldap_friendly(3N)</code> |
| <code>ldap_get_dn(3N)</code> | LDAP DN handling functions |

| | |
|--|---|
| <code>ldap_get_values(3N)</code> | LDAP attribute value handling functions |
| <code>ldap_get_values_len(3N)</code> | See <code>ldap_get_values(3N)</code> |
| <code>ldap_getfilter(3N)</code> | LDAP filter generating functions |
| <code>ldap_getfilter_free(3N)</code> | See <code>ldap_getfilter(3N)</code> |
| <code>ldap_getfirstfilter(3N)</code> | See <code>ldap_getfilter(3N)</code> |
| <code>ldap_getnextfilter(3N)</code> | See <code>ldap_getfilter(3N)</code> |
| <code>ldap_init(3N)</code> | See <code>ldap_open(3N)</code> |
| <code>ldap_init_getfilter(3N)</code> | See <code>ldap_getfilter(3N)</code> |
| <code>ldap_init_getfilter_buf(3N)</code> | See <code>ldap_getfilter(3N)</code> |
| <code>ldap_init_searchprefs(3N)</code> | See <code>ldap_searchprefs(3N)</code> |
| <code>ldap_init_searchprefs_buf(3N)</code> | See <code>ldap_searchprefs(3N)</code> |
| <code>ldap_init_templates(3N)</code> | See <code>ldap_disptmpl(3N)</code> |
| <code>ldap_init_templates_buf(3N)</code> | See <code>ldap_disptmpl(3N)</code> |
| <code>ldap_is_dns_dn(3N)</code> | See <code>ldap_get_dn(3N)</code> |
| <code>ldap_is_ldap_url(3N)</code> | See <code>ldap_url(3N)</code> |
| <code>ldap_modify(3N)</code> | LDAP entry modification functions |
| <code>ldap_modify_ext(3N)</code> | See <code>ldap_modify(3N)</code> |
| <code>ldap_modify_ext_s(3N)</code> | See <code>ldap_modify(3N)</code> |
| <code>ldap_modify_s(3N)</code> | See <code>ldap_modify(3N)</code> |
| <code>ldap_modrdn(3N)</code> | modify LDAP entry RDN |
| <code>ldap_modrdn2(3N)</code> | See <code>ldap_modrdn(3N)</code> |
| <code>ldap_modrdn2_s(3N)</code> | See <code>ldap_modrdn(3N)</code> |
| <code>ldap_modrdn_s(3N)</code> | See <code>ldap_modrdn(3N)</code> |

| | |
|--|---|
| <code>ldap_mods_free(3N)</code> | See <code>ldap_modify(3N)</code> |
| <code>ldap_msgtype(3N)</code> | See <code>ldap_first_message(3N)</code> |
| <code>ldap_next_attribute(3N)</code> | See <code>ldap_first_attribute(3N)</code> |
| <code>ldap_next_disptmpl(3N)</code> | See <code>ldap_disptmpl(3N)</code> |
| <code>ldap_next_entry(3N)</code> | See <code>ldap_first_entry(3N)</code> |
| <code>ldap_next_message(3N)</code> | See <code>ldap_first_message(3N)</code> |
| <code>ldap_next_searchobj(3N)</code> | See <code>ldap_searchprefs(3N)</code> |
| <code>ldap_next_tmplcol(3N)</code> | See <code>ldap_disptmpl(3N)</code> |
| <code>ldap_next_tmplrow(3N)</code> | See <code>ldap_disptmpl(3N)</code> |
| <code>ldap_oc2template(3N)</code> | See <code>ldap_disptmpl(3N)</code> |
| <code>ldap_open(3N)</code> | initialize the LDAP library and open a connection to an LDAP server |
| <code>ldap_parse_extended_result(3N)</code> | See <code>ldap_parse_result(3N)</code> |
| <code>ldap_parse_result(3N)</code> | LDAP message result parser |
| <code>ldap_parse_sasl_bind_result(3N)</code> | See <code>ldap_parse_result(3N)</code> |
| <code>ldap_perror(3N)</code> | See <code>ldap_error(3N)</code> |
| <code>ldap_rename(3N)</code> | See <code>ldap_modrdn(3N)</code> |
| <code>ldap_rename_s(3N)</code> | See <code>ldap_modrdn(3N)</code> |
| <code>ldap_result(3N)</code> | wait for and return LDAP operation result |
| <code>ldap_result2error(3N)</code> | See <code>ldap_error(3N)</code> |
| <code>ldap_sasl_bind(3N)</code> | See <code>ldap_bind(3N)</code> |
| <code>ldap_sasl_bind_s(3N)</code> | See <code>ldap_bind(3N)</code> |

| | |
|--|---|
| <code>ldap_search(3N)</code> | LDAP search operations |
| <code>ldap_search_ext(3N)</code> | See <code>ldap_search(3N)</code> |
| <code>ldap_search_ext_s(3N)</code> | See <code>ldap_search(3N)</code> |
| <code>ldap_search_s(3N)</code> | See <code>ldap_search(3N)</code> |
| <code>ldap_search_st(3N)</code> | See <code>ldap_search(3N)</code> |
| <code>ldap_searchprefs(3N)</code> | LDAP search preference configuration routines |
| <code>ldap_set_cache_options(3N)</code> | See <code>ldap_cache(3N)</code> |
| <code>ldap_set_rebind_proc(3N)</code> | See <code>ldap_bind(3N)</code> |
| <code>ldap_set_string_translators(3N)</code> | See <code>ldap_charset(3N)</code> |
| <code>ldap_simple_bind(3N)</code> | See <code>ldap_bind(3N)</code> |
| <code>ldap_simple_bind_s(3N)</code> | See <code>ldap_bind(3N)</code> |
| <code>ldap_sort(3N)</code> | LDAP entry sorting functions |
| <code>ldap_sort_entries(3N)</code> | See <code>ldap_sort(3N)</code> |
| <code>ldap_sort_strcasecmp(3N)</code> | See <code>ldap_sort(3N)</code> |
| <code>ldap_sort_values(3N)</code> | See <code>ldap_sort(3N)</code> |
| <code>ldap_t61_to_8859(3N)</code> | See <code>ldap_charset(3N)</code> |
| <code>ldap_tmplattrs(3N)</code> | See <code>ldap_disptmpl(3N)</code> |
| <code>ldap_translate_from_t61(3N)</code> | See <code>ldap_charset(3N)</code> |
| <code>ldap_translate_to_t61(3N)</code> | See <code>ldap_charset(3N)</code> |
| <code>ldap_ufn(3N)</code> | LDAP user friendly search functions |
| <code>ldap_ufn_search_c(3N)</code> | See <code>ldap_ufn(3N)</code> |
| <code>ldap_ufn_search_ct(3N)</code> | See <code>ldap_ufn(3N)</code> |
| <code>ldap_ufn_search_s(3N)</code> | See <code>ldap_ufn(3N)</code> |
| <code>ldap_ufn_setfilter(3N)</code> | See <code>ldap_ufn(3N)</code> |

| | |
|---------------------------------------|--|
| <code>ldap_ufn_setprefix(3N)</code> | See <code>ldap_ufn(3N)</code> |
| <code>ldap_ufn_timeout(3N)</code> | See <code>ldap_ufn(3N)</code> |
| <code>ldap_unbind(3N)</code> | See <code>ldap_bind(3N)</code> |
| <code>ldap_unbind_s(3N)</code> | See <code>ldap_bind(3N)</code> |
| <code>ldap_uncache_entry(3N)</code> | See <code>ldap_cache(3N)</code> |
| <code>ldap_uncache_request(3N)</code> | See <code>ldap_cache(3N)</code> |
| <code>ldap_url(3N)</code> | LDAP Uniform Resource Locator functions |
| <code>ldap_url_parse(3N)</code> | See <code>ldap_url(3N)</code> |
| <code>ldap_url_search(3N)</code> | See <code>ldap_url(3N)</code> |
| <code>ldap_url_search_s(3N)</code> | See <code>ldap_url(3N)</code> |
| <code>ldap_url_search_st(3N)</code> | See <code>ldap_url(3N)</code> |
| <code>ldap_vals2html(3N)</code> | See <code>ldap_entry2text(3N)</code> |
| <code>ldap_vals2text(3N)</code> | See <code>ldap_entry2text(3N)</code> |
| <code>ldap_value_free(3N)</code> | See <code>ldap_get_values(3N)</code> |
| <code>ldap_value_free_len(3N)</code> | See <code>ldap_get_values(3N)</code> |
| <code>ldexp(3C)</code> | load exponent of a floating point number |
| <code>ldiv(3C)</code> | See <code>div(3C)</code> |
| <code>leaveok(3X)</code> | See <code>curls_outopts(3X)</code> |
| <code>leaveok(3XC)</code> | See <code>clearok(3XC)</code> |
| <code>lfind(3C)</code> | See <code>lsearch(3C)</code> |
| <code>lfmt(3C)</code> | display error message in standard format and pass to logging and monitoring services |
| <code>lgamma(3M)</code> | log gamma function |

| | |
|---------------------------|---|
| lgamma_r(3M) | See lgamma(3M) |
| libdevinfo(3) | library of device information functions |
| libpthread(3T) | See threads(3T) |
| libthread(3T) | See threads(3T) |
| libthread_db(3T) | library of interfaces for monitoring and manipulating threads-related aspects of multithreaded programs |
| libtnfctl(3X) | library for TNF probe control in a process or the kernel |
| line(3) | See plot(3) |
| link_field(3X) | See form_field_new(3X) |
| link_fieldtype(3X) | See form_fieldtype(3X) |
| linmod(3) | See plot(3) |
| lio_listio(3R) | list directed I/O |
| listen(3N) | listen for connections on a socket |
| listen(3XN) | listen for socket connections and limit the queue of incoming connections |
| llabs(3C) | See abs(3C) |
| lldiv(3C) | See div(3C) |
| lltostr(3C) | See strtol(3C) |
| localeconv(3C) | get numeric formatting information |
| localtime(3C) | See ctime(3C) |
| localtime_r(3C) | See ctime(3C) |
| lockf(3C) | record locking on files |

| | |
|------------------------|---|
| log(3M) | natural logarithm function |
| log10(3M) | base 10 logarithm function |
| log1p(3M) | compute natural logarithm |
| logb(3M) | radix-independent exponent |
| longjmp(3B) | See setjmp(3B) |
| longjmp(3C) | See setjmp(3C) |
| longname(3X) | See curls_termattrs(3X) |
| longname(3XC) | return full terminal type name |
| lrand48(3C) | See drand48(3C) |
| lsearch(3C) | linear search and update |
| madvise(3) | provide advice to VM system |
| maillock(3X) | functions to manage lockfile(s) for user's mailbox |
| mailunlock(3X) | See maillock(3X) |
| major(3C) | See makedev(3C) |
| makecontext(3C) | manipulate user contexts |
| makedev(3C) | manage a device number |
| mallinfo(3X) | See malloc(3X) |
| malloc(3C) | memory allocator |
| malloc(3X) | memory allocator |
| malloc(3t) | See mtmalloc(3t) |
| mallocctl(3t) | See mtmalloc(3t) |
| mallopt(3X) | See malloc(3X) |
| mapmalloc(3X) | memory allocator |

| | |
|---------------------------|--|
| matherr(3M) | math library exception-handling function |
| mblen(3C) | get number of bytes in a character |
| mbrlen(3C) | get number of bytes in a character (restartable) |
| mbrtowc(3C) | convert a character to a wide-character code (restartable) |
| mbsinit(3C) | determine conversion object status |
| mbsrtowcs(3C) | convert a character string to a wide-character string (restartable) |
| mbstowcs(3C) | convert a character string to a wide-character string |
| mbtowc(3C) | convert a character to a wide-character code |
| mctl(3B) | memory management control |
| media_findname(3X) | convert a supplied name into an absolute pathname that can be used to access removable media |
| media_getattr(3X) | get and set media attributes |
| media_getid(3X) | return the id of a piece of media |
| media_setattr(3X) | See media_getattr(3X) |
| memalign(3C) | See malloc(3C) |
| memalign(3X) | See watchmalloc(3X) |
| memccpy(3C) | See memory(3C) |
| memchr(3C) | See memory(3C) |
| memcmp(3C) | See memory(3C) |

| | |
|------------------------------|--|
| memcpy(3C) | See memory(3C) |
| memmove(3C) | See memory(3C) |
| memory(3C) | memory operations |
| memset(3C) | See memory(3C) |
| menu_attributes(3X) | control menus display attributes |
| menu_back(3X) | See menu_attributes(3X) |
| menu_cursor(3X) | correctly position a menus cursor |
| menu_driver(3X) | command processor for the menus subsystem |
| menu_fore(3X) | See menu_attributes(3X) |
| menu_format(3X) | set and get maximum numbers of rows and columns in menus |
| menu_grey(3X) | See menu_attributes(3X) |
| menu_hook(3X) | assign application-specific routines for automatic invocation by menus |
| menu_init(3X) | See menu_hook(3X) |
| menu_item_current(3X) | set and get current menus items |
| menu_item_name(3X) | get menus item name and description |
| menu_item_new(3X) | create and destroy menus items |
| menu_item_opts(3X) | menus item option routines |
| menu_item_userptr(3X) | associate application data with menus items |
| menu_item_value(3X) | set and get menus item values |
| menu_item_visible(3X) | tell if menus item is visible |

| | |
|--------------------------------|---|
| <code>menu_items(3X)</code> | connect and disconnect items to and from menus |
| <code>menu_mark(3X)</code> | menus mark string routines |
| <code>menu_new(3X)</code> | create and destroy menus |
| <code>menu_opts(3X)</code> | menus option routines |
| <code>menu_opts_off(3X)</code> | See <code>menu_opts(3X)</code> |
| <code>menu_opts_on(3X)</code> | See <code>menu_opts(3X)</code> |
| <code>menu_pad(3X)</code> | See <code>menu_attributes(3X)</code> |
| <code>menu_pattern(3X)</code> | set and get menus pattern match buffer |
| <code>menu_post(3X)</code> | write or erase menus from associated subwindows |
| <code>menu_sub(3X)</code> | See <code>menu_win(3X)</code> |
| <code>menu_term(3X)</code> | See <code>menu_hook(3X)</code> |
| <code>menu_userptr(3X)</code> | associate application data with menus |
| <code>menu_win(3X)</code> | menus window and subwindow association routines |
| <code>menus(3X)</code> | character based menus package |
| <code>meta(3X)</code> | See <code>curs_inopts(3X)</code> |
| <code>meta(3XC)</code> | enable/disable meta keys |
| <code>minor(3C)</code> | See <code>makedev(3C)</code> |
| <code>mkdirp(3G)</code> | create, remove directories in a path |
| <code>mkfifo(3C)</code> | create a new FIFO |
| <code>mkstemp(3C)</code> | make a unique file name |
| <code>mktemp(3C)</code> | make a unique file name |

| | |
|-----------------------|--|
| mktime(3C) | converts a tm structure to a calendar time |
| mlock(3C) | lock or unlock pages in memory |
| mlockall(3C) | lock or unlock address space |
| modf(3C) | decompose floating-point number |
| modff(3C) | See modf(3C) |
| monitor(3C) | prepare process execution profile |
| move(3) | See plot(3) |
| move(3X) | See curs_move(3X) |
| move(3XC) | move cursor in window |
| move_field(3X) | See form_field(3X) |
| move_panel(3X) | See panel_move(3X) |
| movenextch(3X) | See curs_alecompat(3X) |
| moveprevch(3X) | See curs_alecompat(3X) |
| mp(3M) | multiple precision integer arithmetic |
| mp_gcd(3M) | See mp(3M) |
| mp_itom(3M) | See mp(3M) |
| mp_madd(3M) | See mp(3M) |
| mp_mcmp(3M) | See mp(3M) |
| mp_mdiv(3M) | See mp(3M) |
| mp_mfree(3M) | See mp(3M) |
| mp_min(3M) | See mp(3M) |
| mp_mout(3M) | See mp(3M) |
| mp_msub(3M) | See mp(3M) |

| | |
|--------------------------|---|
| mp_mtox(3M) | See mp(3M) |
| mp_mult(3M) | See mp(3M) |
| mp_pow(3M) | See mp(3M) |
| mp_rpow(3M) | See mp(3M) |
| mp_xtom(3M) | See mp(3M) |
| mq_close(3R) | close a message queue |
| mq_getattr(3R) | get message queue attributes |
| mq_notify(3R) | notify process (or thread) that a message is available on a queue |
| mq_open(3R) | open a message queue |
| mq_receive(3R) | receive a message from a message queue |
| mq_send(3R) | send a message to a message queue |
| mq_setattr(3R) | set/get message queue attributes |
| mq_unlink(3R) | remove a message queue |
| rand48(3C) | See drand48(3C) |
| msync(3C) | synchronize memory with physical storage |
| mtmalloc(3t) | MT hot memory allocator |
| munlock(3C) | See mlock(3C) |
| munlockall(3C) | See mlockall(3C) |
| mutex(3T) | concepts relating to mutual exclusion locks |
| mutex_destroy(3T) | See mutex_init(3T) |
| mutex_init(3T) | mutual exclusion locks |

| | |
|---------------------------------|--------------------------------------|
| <code>mutex_lock(3T)</code> | See <code>mutex_init(3T)</code> |
| <code>mutex_trylock(3T)</code> | See <code>mutex_init(3T)</code> |
| <code>mutex_unlock(3T)</code> | See <code>mutex_init(3T)</code> |
| <code>mvadd_wch(3XC)</code> | See <code>add_wch(3XC)</code> |
| <code>mvadd_wchnstr(3XC)</code> | See <code>add_wchnstr(3XC)</code> |
| <code>mvadd_wchstr(3XC)</code> | See <code>add_wchnstr(3XC)</code> |
| <code>mvaddch(3X)</code> | See <code> curs_addch(3X)</code> |
| <code>mvaddch(3XC)</code> | See <code>addch(3XC)</code> |
| <code>mvaddchnstr(3X)</code> | See <code> curs_addchnstr(3X)</code> |
| <code>mvaddchnstr(3XC)</code> | See <code>addchnstr(3XC)</code> |
| <code>mvaddchstr(3X)</code> | See <code> curs_addchstr(3X)</code> |
| <code>mvaddchstr(3XC)</code> | See <code>addchstr(3XC)</code> |
| <code>mvaddnstr(3X)</code> | See <code> curs_addstr(3X)</code> |
| <code>mvaddnstr(3XC)</code> | See <code>addnstr(3XC)</code> |
| <code>mvaddnwstr(3X)</code> | See <code> curs_addwstr(3X)</code> |
| <code>mvaddnwstr(3XC)</code> | See <code>addnwstr(3XC)</code> |
| <code>mvaddstr(3X)</code> | See <code> curs_addstr(3X)</code> |
| <code>mvaddstr(3XC)</code> | See <code>addnstr(3XC)</code> |
| <code>mvaddwch(3X)</code> | See <code> curs_addwch(3X)</code> |
| <code>mvaddwchnstr(3X)</code> | See <code> curs_addwchstr(3X)</code> |
| <code>mvaddwchstr(3X)</code> | See <code> curs_addwchstr(3X)</code> |
| <code>mvaddwstr(3X)</code> | See <code> curs_addwstr(3X)</code> |
| <code>mvaddwstr(3XC)</code> | See <code>addnwstr(3XC)</code> |
| <code>mvchgat(3XC)</code> | See <code>chgat(3XC)</code> |
| <code>mvcur(3X)</code> | See <code> curs_terminfo(3X)</code> |

| | |
|--------------------------|--|
| mvcur(3XC) | move the cursor |
| mvdelch(3X) | See curs_delch(3X) |
| mvdelch(3XC) | See delch(3XC) |
| mvderwin(3X) | See curs_window(3X) |
| mvderwin(3XC) | map area of parent window to subwindow |
| mvget_wch(3XC) | See get_wch(3XC) |
| mvget_wstr(3XC) | See getn_wstr(3XC) |
| mvgetch(3X) | See curs_getch(3X) |
| mvgetch(3XC) | See getch(3XC) |
| mvgetn_wstr(3XC) | See getn_wstr(3XC) |
| mvgetnstr(3XC) | See getnstr(3XC) |
| mvgetnwstr(3X) | See curs_getwstr(3X) |
| mvgetstr(3X) | See curs_getstr(3X) |
| mvgetstr(3XC) | See getnstr(3XC) |
| mvgetwch(3X) | See curs_getwch(3X) |
| mvgetwstr(3X) | See curs_getwstr(3X) |
| mvhline(3XC) | See hline(3XC) |
| mvhline_set(3XC) | See hline_set(3XC) |
| mvin_wch(3XC) | See in_wch(3XC) |
| mvin_wchnstr(3XC) | See in_wchnstr(3XC) |
| mvin_wchstr(3XC) | See in_wchnstr(3XC) |
| mvinch(3X) | See curs_inch(3X) |
| mvinch(3XC) | See inch(3XC) |
| mvinchnstr(3X) | See curs_inchstr(3X) |

| | |
|-------------------------------|-------------------------------------|
| <code>mvinchnstr(3XC)</code> | See <code>inchnstr(3XC)</code> |
| <code>mvinchstr(3X)</code> | See <code> curs_inchstr(3X)</code> |
| <code>mvinchstr(3XC)</code> | See <code>inchnstr(3XC)</code> |
| <code>mvinnstr(3X)</code> | See <code> curs_instr(3X)</code> |
| <code>mvinnstr(3XC)</code> | See <code>innstr(3XC)</code> |
| <code>mvinnwstr(3X)</code> | See <code> curs_inwstr(3X)</code> |
| <code>mvinnwstr(3XC)</code> | See <code>innwstr(3XC)</code> |
| <code>mvins_nwstr(3XC)</code> | See <code>ins_nwstr(3XC)</code> |
| <code>mvins_wch(3XC)</code> | See <code>ins_wch(3XC)</code> |
| <code>mvins_wstr(3XC)</code> | See <code>ins_nwstr(3XC)</code> |
| <code>mvinsch(3X)</code> | See <code> curs_insch(3X)</code> |
| <code>mvinsch(3XC)</code> | See <code>insch(3XC)</code> |
| <code>mvinsnstr(3X)</code> | See <code> curs_insstr(3X)</code> |
| <code>mvinsnstr(3XC)</code> | See <code>insnstr(3XC)</code> |
| <code>mvinsnwstr(3X)</code> | See <code> curs_inswstr(3X)</code> |
| <code>mvinsstr(3X)</code> | See <code> curs_insstr(3X)</code> |
| <code>mvinsstr(3XC)</code> | See <code>insnstr(3XC)</code> |
| <code>mvinstr(3X)</code> | See <code> curs_instr(3X)</code> |
| <code>mvinstr(3XC)</code> | See <code>innstr(3XC)</code> |
| <code>mvinswch(3X)</code> | See <code> curs_inswch(3X)</code> |
| <code>mvinswstr(3X)</code> | See <code> curs_inswstr(3X)</code> |
| <code>mvinwch(3X)</code> | See <code> curs_inwch(3X)</code> |
| <code>mvinwchnstr(3X)</code> | See <code> curs_inwchstr(3X)</code> |
| <code>mvinwchstr(3X)</code> | See <code> curs_inwchstr(3X)</code> |
| <code>mvinwstr(3X)</code> | See <code> curs_inwstr(3X)</code> |

| | |
|----------------------------------|-------------------------------------|
| <code>mvwinstr(3XC)</code> | See <code>innwstr(3XC)</code> |
| <code>mvprintw(3X)</code> | See <code>curs_printw(3X)</code> |
| <code>mvprintw(3XC)</code> | write formatted output to window |
| <code>mvscanw(3X)</code> | See <code>curs_scanw(3X)</code> |
| <code>mvscanw(3XC)</code> | read formatted input from window |
| <code>mvvline(3XC)</code> | See <code>hline(3XC)</code> |
| <code>mvvline_set(3XC)</code> | See <code>hline_set(3XC)</code> |
| <code>mvwadd_wch(3XC)</code> | See <code>add_wch(3XC)</code> |
| <code>mvwadd_wchnstr(3XC)</code> | See <code>add_wchnstr(3XC)</code> |
| <code>mvwadd_wchstr(3XC)</code> | See <code>add_wchnstr(3XC)</code> |
| <code>mvwaddch(3X)</code> | See <code>curs_addch(3X)</code> |
| <code>mvwaddch(3XC)</code> | See <code>addch(3XC)</code> |
| <code>mvwaddchnstr(3X)</code> | See <code>curs_addchnstr(3X)</code> |
| <code>mvwaddchnstr(3XC)</code> | See <code>addchnstr(3XC)</code> |
| <code>mvwaddchstr(3X)</code> | See <code>curs_addchstr(3X)</code> |
| <code>mvwaddchstr(3XC)</code> | See <code>addchstr(3XC)</code> |
| <code>mvwaddnstr(3X)</code> | See <code>curs_addstr(3X)</code> |
| <code>mvwaddnstr(3XC)</code> | See <code>addnstr(3XC)</code> |
| <code>mvwaddnwstr(3X)</code> | See <code>curs_addwstr(3X)</code> |
| <code>mvwaddnwstr(3XC)</code> | See <code>addnwstr(3XC)</code> |
| <code>mvwaddstr(3X)</code> | See <code>curs_addstr(3X)</code> |
| <code>mvwaddstr(3XC)</code> | See <code>addnstr(3XC)</code> |
| <code>mvwaddwch(3X)</code> | See <code>curs_addwch(3X)</code> |
| <code>mvwaddwchnstr(3X)</code> | See <code>curs_addwchstr(3X)</code> |

| | |
|---------------------------------|--------------------------------------|
| <code>mvwaddwchstr(3X)</code> | See <code> curs_addwchstr(3X)</code> |
| <code>mvwaddwstr(3X)</code> | See <code> curs_addwstr(3X)</code> |
| <code>mvwaddwstr(3XC)</code> | See <code> addnwstr(3XC)</code> |
| <code>mvwchgat(3XC)</code> | See <code> chgat(3XC)</code> |
| <code>mvwdelch(3X)</code> | See <code> curs_delch(3X)</code> |
| <code>mvwdelch(3XC)</code> | See <code> delch(3XC)</code> |
| <code>mvwget_wch(3XC)</code> | See <code> get_wch(3XC)</code> |
| <code>mvwget_wstr(3XC)</code> | See <code> getn_wstr(3XC)</code> |
| <code>mvwgetch(3X)</code> | See <code> curs_getch(3X)</code> |
| <code>mvwgetch(3XC)</code> | See <code> getch(3XC)</code> |
| <code>mvwgetn_wstr(3XC)</code> | See <code> getn_wstr(3XC)</code> |
| <code>mvwgetnstr(3XC)</code> | See <code> getnstr(3XC)</code> |
| <code>mvwgetnwstr(3X)</code> | See <code> curs_getwstr(3X)</code> |
| <code>mvwgetstr(3X)</code> | See <code> curs_getstr(3X)</code> |
| <code>mvwgetstr(3XC)</code> | See <code> getnstr(3XC)</code> |
| <code>mvwgetwch(3X)</code> | See <code> curs_getwch(3X)</code> |
| <code>mvwgetwstr(3X)</code> | See <code> curs_getwstr(3X)</code> |
| <code>mvwhline(3XC)</code> | See <code> hline(3XC)</code> |
| <code>mvwhline_set(3XC)</code> | See <code> hline_set(3XC)</code> |
| <code>mvwin(3X)</code> | See <code> curs_window(3X)</code> |
| <code>mvwin(3XC)</code> | move window |
| <code>mvwin_wch(3XC)</code> | See <code> in_wch(3XC)</code> |
| <code>mvwin_wchnstr(3XC)</code> | See <code> in_wchnstr(3XC)</code> |
| <code>mvwin_wchstr(3XC)</code> | See <code> in_wchnstr(3XC)</code> |
| <code>mvwinch(3X)</code> | See <code> curs_inch(3X)</code> |

| | |
|--------------------------------|-------------------------------------|
| <code>mvwinch(3XC)</code> | See <code>inch(3XC)</code> |
| <code>mvwinchnstr(3X)</code> | See <code> curs_inchstr(3X)</code> |
| <code>mvwinchnstr(3XC)</code> | See <code> inchnstr(3XC)</code> |
| <code>mvwinchstr(3X)</code> | See <code> curs_inchstr(3X)</code> |
| <code>mvwinchstr(3XC)</code> | See <code> inchnstr(3XC)</code> |
| <code>mvwinnstr(3X)</code> | See <code> curs_instr(3X)</code> |
| <code>mvwinnstr(3XC)</code> | See <code> innstr(3XC)</code> |
| <code>mvwinnwstr(3X)</code> | See <code> curs_inwstr(3X)</code> |
| <code>mvwinnwstr(3XC)</code> | See <code> innwstr(3XC)</code> |
| <code>mvwins_nstr(3XC)</code> | See <code> ins_nwstr(3XC)</code> |
| <code>mvwins_nwstr(3XC)</code> | See <code> ins_nwstr(3XC)</code> |
| <code>mvwins_wch(3XC)</code> | See <code> ins_wch(3XC)</code> |
| <code>mvwinsch(3X)</code> | See <code> curs_insch(3X)</code> |
| <code>mvwinsch(3XC)</code> | See <code> insch(3XC)</code> |
| <code>mvwinsnstr(3X)</code> | See <code> curs_insstr(3X)</code> |
| <code>mvwinsnstr(3XC)</code> | See <code> insnstr(3XC)</code> |
| <code>mvwinsnwstr(3X)</code> | See <code> curs_inswstr(3X)</code> |
| <code>mvwinsstr(3X)</code> | See <code> curs_insstr(3X)</code> |
| <code>mvwinsstr(3XC)</code> | See <code> insnstr(3XC)</code> |
| <code>mvwinstr(3X)</code> | See <code> curs_instr(3X)</code> |
| <code>mvwinstr(3XC)</code> | See <code> innstr(3XC)</code> |
| <code>mvwinswch(3X)</code> | See <code> curs_inswch(3X)</code> |
| <code>mvwinswstr(3X)</code> | See <code> curs_inswstr(3X)</code> |
| <code>mvwinwch(3X)</code> | See <code> curs_inwch(3X)</code> |
| <code>mvwinwchnstr(3X)</code> | See <code> curs_inwchstr(3X)</code> |

| | |
|-----------------------------------|---|
| <code>mvwinwchstr(3X)</code> | See <code> curs_inwchstr(3X)</code> |
| <code>mvwinwstr(3X)</code> | See <code> curs_inwstr(3X)</code> |
| <code>mvwinwstr(3XC)</code> | See <code> innwstr(3XC)</code> |
| <code>mvwprintw(3X)</code> | See <code> curs_printw(3X)</code> |
| <code>mvwprintw(3XC)</code> | See <code> mvprintw(3XC)</code> |
| <code>mvwscanw(3X)</code> | See <code> curs_scanw(3X)</code> |
| <code>mvwscanw(3XC)</code> | See <code> mvscanw(3XC)</code> |
| <code>mvwvline(3XC)</code> | See <code> hline(3XC)</code> |
| <code>mvwvline_set(3XC)</code> | See <code> hline_set(3XC)</code> |
| <code>nanosleep(3R)</code> | high resolution sleep |
| <code>napms(3X)</code> | See <code> curs_kernel(3X)</code> |
| <code>napms(3XC)</code> | sleep process for a specified length of time |
| <code>nc_perror(3N)</code> | See <code> getnetconfig(3N)</code> |
| <code>nc_serror(3N)</code> | See <code> getnetconfig(3N)</code> |
| <code>netdir(3N)</code> | generic transport name-to-address translation |
| <code>netdir_free(3N)</code> | See <code> netdir(3N)</code> |
| <code>netdir_getbyaddr(3N)</code> | See <code> netdir(3N)</code> |
| <code>netdir_getbyname(3N)</code> | See <code> netdir(3N)</code> |
| <code>netdir_mergeaddr(3N)</code> | See <code> netdir(3N)</code> |
| <code>netdir_options(3N)</code> | See <code> netdir(3N)</code> |
| <code>netdir_perror(3N)</code> | See <code> netdir(3N)</code> |
| <code>netdir_serror(3N)</code> | See <code> netdir(3N)</code> |
| <code>netname2host(3N)</code> | See <code> secure_rpc(3N)</code> |
| <code>netname2user(3N)</code> | See <code> secure_rpc(3N)</code> |

| | |
|-----------------------------|---|
| new_field(3X) | See form_field_new(3X) |
| new_fieldtype(3X) | See form_fieldtype(3X) |
| new_form(3X) | See form_new(3X) |
| new_item(3X) | See menu_item_new(3X) |
| new_menu(3X) | See menu_new(3X) |
| new_page(3X) | See form_new_page(3X) |
| new_panel(3X) | See panel_new(3X) |
| newpad(3X) | See curs_pad(3X) |
| newpad(3XC) | create or refresh a pad or subpad |
| newterm(3X) | See curs_initscr(3X) |
| newterm(3XC) | See initscr(3XC) |
| newwin(3X) | See curs_window(3X) |
| newwin(3XC) | See derwin(3XC) |
| nextafter(3M) | next representable double-precision floating-point number |
| nextkey(3B) | See dbm(3B) |
| nftw(3C) | See ftw(3C) |
| nice(3B) | change priority of a process |
| nis_add(3N) | See nis_names(3N) |
| nis_add_entry(3N) | See nis_tables(3N) |
| nis_addmember(3N) | See nis_groups(3N) |
| nis_checkpoint(3N) | See nis_ping(3N) |
| nis_clone_object(3N) | See nis_subr(3N) |
| nis_creategroup(3N) | See nis_groups(3N) |

| | |
|--------------------------------------|--------------------------------------|
| <code>nis_db(3N)</code> | NIS+ Database access functions |
| <code>nis_destroy_object(3N)</code> | See <code>nis_subr(3N)</code> |
| <code>nis_destroygroup(3N)</code> | See <code>nis_groups(3N)</code> |
| <code>nis_dir_cmp(3N)</code> | See <code>nis_subr(3N)</code> |
| <code>nis_domain_of(3N)</code> | See <code>nis_subr(3N)</code> |
| <code>nis_error(3N)</code> | display NIS+ error messages |
| <code>nis_first_entry(3N)</code> | See <code>nis_tables(3N)</code> |
| <code>nis_freenames(3N)</code> | See <code>nis_subr(3N)</code> |
| <code>nis_freeresult(3N)</code> | See <code>nis_names(3N)</code> |
| <code>nis_freeservlist(3N)</code> | See <code>nis_server(3N)</code> |
| <code>nis_freetags(3N)</code> | See <code>nis_server(3N)</code> |
| <code>nis_getnames(3N)</code> | See <code>nis_subr(3N)</code> |
| <code>nis_getservlist(3N)</code> | See <code>nis_server(3N)</code> |
| <code>nis_groups(3N)</code> | NIS+ group manipulation functions |
| <code>nis_ismember(3N)</code> | See <code>nis_groups(3N)</code> |
| <code>nis_leaf_of(3N)</code> | See <code>nis_subr(3N)</code> |
| <code>nis_lerror(3N)</code> | See <code>nis_error(3N)</code> |
| <code>nis_list(3N)</code> | See <code>nis_tables(3N)</code> |
| <code>nis_local_directory(3N)</code> | See <code>nis_local_names(3N)</code> |
| <code>nis_local_group(3N)</code> | See <code>nis_local_names(3N)</code> |
| <code>nis_local_host(3N)</code> | See <code>nis_local_names(3N)</code> |
| <code>nis_local_names(3N)</code> | NIS+ local names |
| <code>nis_local_principal(3N)</code> | See <code>nis_local_names(3N)</code> |
| <code>nis_lookup(3N)</code> | See <code>nis_names(3N)</code> |

| | |
|--|--|
| <code>nis_mkdir(3N)</code> | See <code>nis_server(3N)</code> |
| <code>nis_modify(3N)</code> | See <code>nis_names(3N)</code> |
| <code>nis_modify_entry(3N)</code> | See <code>nis_tables(3N)</code> |
| <code>nis_name_of(3N)</code> | See <code>nis_subr(3N)</code> |
| <code>nis_names(3N)</code> | NIS+ namespace functions |
| <code>nis_next_entry(3N)</code> | See <code>nis_tables(3N)</code> |
| <code>nis_objects(3N)</code> | NIS+ object formats |
| <code>nis_perror(3N)</code> | See <code>nis_error(3N)</code> |
| <code>nis_ping(3N)</code> | misc NIS+ log administration functions |
| <code>nis_print_group_entry(3N)</code> | See <code>nis_groups(3N)</code> |
| <code>nis_print_object(3N)</code> | See <code>nis_subr(3N)</code> |
| <code>nis_remove(3N)</code> | See <code>nis_names(3N)</code> |
| <code>nis_remove_entry(3N)</code> | See <code>nis_tables(3N)</code> |
| <code>nis_removemember(3N)</code> | See <code>nis_groups(3N)</code> |
| <code>nis_rmdir(3N)</code> | See <code>nis_server(3N)</code> |
| <code>nis_server(3N)</code> | miscellaneous NIS+ functions |
| <code>nis_servstate(3N)</code> | See <code>nis_server(3N)</code> |
| <code>nis_sperrno(3N)</code> | See <code>nis_error(3N)</code> |
| <code>nis_sperror(3N)</code> | See <code>nis_error(3N)</code> |
| <code>nis_sperror_r(3N)</code> | See <code>nis_error(3N)</code> |
| <code>nis_stats(3N)</code> | See <code>nis_server(3N)</code> |
| <code>nis_subr(3N)</code> | NIS+ subroutines |
| <code>nis_tables(3N)</code> | NIS+ table functions |
| <code>nis_verifygroup(3N)</code> | See <code>nis_groups(3N)</code> |

| | |
|------------------------|--|
| nl(3X) | See curs_ouptos(3X) |
| nl(3XC) | enable/disable newline control |
| nl_langinfo(3C) | language information |
| nlist(3B) | get entries from symbol table |
| nlist(3E) | get entries from name list |
| nlsgetcall(3N) | get client's data passed via the listener |
| nlsprovider(3N) | get name of transport provider |
| nlsrequest(3N) | format and send listener service request message |
| nocbreak(3X) | See curs_inoptos(3X) |
| nocbreak(3XC) | See cbreak(3XC) |
| nodelay(3X) | See curs_inoptos(3X) |
| nodelay(3XC) | set blocking or non-blocking read |
| noecho(3X) | See curs_inoptos(3X) |
| noecho(3XC) | See echo(3XC) |
| nonl(3X) | See curs_ouptos(3X) |
| nonl(3XC) | See nl(3XC) |
| noqiflush(3X) | See curs_inoptos(3X) |
| noqiflush(3XC) | control flush of input and output on interrupt |
| noraw(3X) | See curs_inoptos(3X) |
| noraw(3XC) | See cbreak(3XC) |
| notimeout(3X) | See curs_inoptos(3X) |
| notimeout(3XC) | set timed blocking or non-blocking read |

| | |
|----------------------------------|---|
| <code>nrand48(3C)</code> | See <code>drand48(3C)</code> |
| <code>ntohl(3N)</code> | See <code>byteorder(3N)</code> |
| <code>ntohl(3XN)</code> | See <code>htonl(3XN)</code> |
| <code>ntohs(3N)</code> | See <code>byteorder(3N)</code> |
| <code>ntohs(3XN)</code> | See <code>htonl(3XN)</code> |
| <code>offsetof(3C)</code> | offset of structure member |
| <code>opendir(3C)</code> | open directory |
| <code>openlog(3)</code> | See <code>syslog(3)</code> |
| <code>openpl(3)</code> | See <code>plot(3)</code> |
| <code>openvt(3)</code> | See <code>plot(3)</code> |
| <code>overlay(3X)</code> | See <code>curs_overlay(3X)</code> |
| <code>overlay(3XC)</code> | overlap or overwrite windows |
| <code>overwrite(3X)</code> | See <code>curs_overlay(3X)</code> |
| <code>overwrite(3XC)</code> | See <code>overlay(3XC)</code> |
| <code>p2close(3G)</code> | See <code>p2open(3G)</code> |
| <code>p2open(3G)</code> | open, close pipes to and from a command |
| <code>pair_content(3X)</code> | See <code>curs_color(3X)</code> |
| <code>pair_content(3XC)</code> | See <code>can_change_color(3XC)</code> |
| <code>pam(3)</code> | PAM (Pluggable Authentication Module) |
| <code>pam_acct_mgmt(3)</code> | perform PAM account validation procedures |
| <code>pam_authenticate(3)</code> | perform authentication within the PAM framework |

| | |
|-------------------------------------|---|
| <code>pam_chauthtok(3)</code> | perform password related functions within the PAM framework |
| <code>pam_close_session(3)</code> | See <code>pam_open_session(3)</code> |
| <code>pam_end(3)</code> | See <code>pam_start(3)</code> |
| <code>pam_get_data(3)</code> | See <code>pam_set_data(3)</code> |
| <code>pam_get_item(3)</code> | See <code>pam_set_item(3)</code> |
| <code>pam_get_user(3)</code> | PAM routine to retrieve user name |
| <code>pam_getenv(3)</code> | returns the value for a PAM environment name |
| <code>pam_getenvlist(3)</code> | returns a list of all the PAM environment variables |
| <code>pam_open_session(3)</code> | perform PAM session creation and termination operations |
| <code>pam_putenv(3)</code> | change or add a value to the PAM environment |
| <code>pam_set_data(3)</code> | PAM routines to maintain module specific state |
| <code>pam_set_item(3)</code> | authentication information routines for PAM |
| <code>pam_setcred(3)</code> | modify/delete user credentials for an authentication service |
| <code>pam_sm(3)</code> | PAM Service Module APIs |
| <code>pam_sm_acct_mgmt(3)</code> | service provider implementation for <code>pam_acct_mgmt</code> |
| <code>pam_sm_authenticate(3)</code> | service provider implementation for <code>pam_authenticate</code> |

| | |
|--------------------------------------|--|
| <code>pam_sm_chauthtok(3)</code> | service provider implementation for <code>pam_chauthtok</code> |
| <code>pam_sm_close_session(3)</code> | See <code>pam_sm_open_session(3)</code> |
| <code>pam_sm_open_session(3)</code> | service provider implementation for <code>pam_open_session</code> and <code>pam_close_session</code> |
| <code>pam_sm_setcred(3)</code> | service provider implementation for <code>pam_setcred</code> |
| <code>pam_start(3)</code> | authentication transaction routines for PAM |
| <code>pam_strerror(3)</code> | get PAM error message string |
| <code>panel_above(3X)</code> | panels deck traversal primitives |
| <code>panel_below(3X)</code> | See <code>panel_above(3X)</code> |
| <code>panel_hidden(3X)</code> | See <code>panel_show(3X)</code> |
| <code>panel_move(3X)</code> | move a panels window on the virtual screen |
| <code>panel_new(3X)</code> | create and destroy panels |
| <code>panel_show(3X)</code> | panels deck manipulation routines |
| <code>panel_top(3X)</code> | panels deck manipulation routines |
| <code>panel_update(3X)</code> | panels virtual screen refresh routine |
| <code>panel_userptr(3X)</code> | associate application data with a panels panel |
| <code>panel_window(3X)</code> | get or set the current window of a panels panel |

| | |
|----------------------------|---|
| panels(3X) | character based panels package |
| pathfind(3G) | search for named file in named directories |
| pclose(3S) | See popen(3S) |
| pecho_wchar(3XC) | See pechochar(3XC) |
| pechochar(3X) | See curs_pad(3X) |
| pechochar(3XC) | add character and refresh window |
| pechowchar(3X) | See curs_pad(3X) |
| perror(3C) | print system error messages |
| pfmt(3C) | display error message in standard format |
| plock(3C) | lock or unlock into memory process, text, or data |
| plot(3) | graphics interface |
| pmap_getmaps(3N) | See rpc_soc(3N) |
| pmap_getport(3N) | See rpc_soc(3N) |
| pmap_rmtcall(3N) | See rpc_soc(3N) |
| pmap_set(3N) | See rpc_soc(3N) |
| pmap_unset(3N) | See rpc_soc(3N) |
| pnoutrefresh(3X) | See curs_pad(3X) |
| pnoutrefresh(3XC) | See newpad(3XC) |
| point(3) | See plot(3) |
| popen(3S) | initiate a pipe to or from a process |
| pos_form_cursor(3X) | See form_cursor(3X) |
| pos_menu_cursor(3X) | See menu_cursor(3X) |

| | |
|----------------------------------|---|
| <code>post_form(3X)</code> | See <code>form_post(3X)</code> |
| <code>post_menu(3X)</code> | See <code>menu_post(3X)</code> |
| <code>pow(3M)</code> | power function |
| <code>prefresh(3X)</code> | See <code>curs_pad(3X)</code> |
| <code>prefresh(3XC)</code> | See <code>newpad(3XC)</code> |
| <code>printf(3B)</code> | formatted output conversion |
| <code>printf(3S)</code> | print formatted output |
| <code>printw(3X)</code> | See <code>curs_printw(3X)</code> |
| <code>printw(3XC)</code> | See <code>mvprintw(3XC)</code> |
| <code>proc_service(3T)</code> | process service interfaces |
| <code>ps_kill(3T)</code> | See <code>ps_pstop(3T)</code> |
| <code>ps_lcontinue(3T)</code> | See <code>ps_pstop(3T)</code> |
| <code>ps_lgetfpregs(3T)</code> | See <code>ps_lgetregs(3T)</code> |
| <code>ps_lgetregs(3T)</code> | routines that access the target process register in <code>libthread_db</code> |
| <code>ps_lgetxregs(3T)</code> | See <code>ps_lgetregs(3T)</code> |
| <code>ps_lgetxregsize(3T)</code> | See <code>ps_lgetregs(3T)</code> |
| <code>ps_lrolltoaddr(3T)</code> | See <code>ps_pstop(3T)</code> |
| <code>ps_lsetfpregs(3T)</code> | See <code>ps_lgetregs(3T)</code> |
| <code>ps_lsetregs(3T)</code> | See <code>ps_lgetregs(3T)</code> |
| <code>ps_lsetxregs(3T)</code> | See <code>ps_lgetregs(3T)</code> |
| <code>ps_lstop(3T)</code> | See <code>ps_pstop(3T)</code> |
| <code>ps_pcontinue(3T)</code> | See <code>ps_pstop(3T)</code> |
| <code>ps_pdread(3T)</code> | See <code>ps_pread(3T)</code> |
| <code>ps_pdwrite(3T)</code> | See <code>ps_pread(3T)</code> |

| | |
|---|---|
| <code>ps_pglobal_lookup(3T)</code> | look up a symbol in the symbol table of the load object in the target process |
| <code>ps_pglobal_sym(3T)</code> | See <code>ps_pglobal_lookup(3T)</code> |
| <code>ps_pread(3T)</code> | interfaces in <code>libthread_db</code> that target process memory access |
| <code>ps_pstop(3T)</code> | process and LWP control in <code>libthread_db</code> |
| <code>ps_ptread(3T)</code> | See <code>ps_pread(3T)</code> |
| <code>ps_ptwrite(3T)</code> | See <code>ps_pread(3T)</code> |
| <code>ps_pwrite(3T)</code> | See <code>ps_pread(3T)</code> |
| <code>psiginfo(3C)</code> | See <code>psignal(3C)</code> |
| <code>psignal(3B)</code> | system signal messages |
| <code>psignal(3C)</code> | system signal messages |
| <code>pthread_atfork(3T)</code> | register fork handlers |
| <code>pthread_attr_destroy(3T)</code> | See <code>pthread_attr_init(3T)</code> |
| <code>pthread_attr_getdetachstate(3T)</code> | get or set detachstate attribute |
| <code>pthread_attr_getguardsize(3T)</code> | get or set the thread guardsize attribute |
| <code>pthread_attr_getinheritsched(3T)</code> | get or set inheritsched attribute |
| <code>pthread_attr_getschedparam(3T)</code> | get or set schedparam attribute |
| <code>pthread_attr_getschedpolicy(3T)</code> | get or set schedpolicy attribute |
| <code>pthread_attr_getscope(3T)</code> | get or set contentionscope attribute |
| <code>pthread_attr_getstackaddr(3T)</code> | get or set stackaddr attribute |
| <code>pthread_attr_getstacksize(3T)</code> | get or set stacksize attribute |

| | |
|---|---|
| <code>pthread_attr_init(3T)</code> | initialize or destroy threads attribute object |
| <code>pthread_attr_setdetachstate(3T)</code> | See <code>pthread_attr_getdetachstate(3T)</code> |
| <code>pthread_attr_setguardsize(3T)</code> | See <code>pthread_attr_getguardsize(3T)</code> |
| <code>pthread_attr_setinheritsched(3T)</code> | See <code>pthread_attr_getinheritsched(3T)</code> |
| <code>pthread_attr_setschedparam(3T)</code> | See <code>pthread_attr_getschedparam(3T)</code> |
| <code>pthread_attr_setschedpolicy(3T)</code> | See <code>pthread_attr_getschedpolicy(3T)</code> |
| <code>pthread_attr_setscope(3T)</code> | See <code>pthread_attr_getscope(3T)</code> |
| <code>pthread_attr_setstackaddr(3T)</code> | See <code>pthread_attr_getstackaddr(3T)</code> |
| <code>pthread_attr_setstacksize(3T)</code> | See <code>pthread_attr_getstacksize(3T)</code> |
| <code>pthread_cancel(3T)</code> | cancel execution of a thread |
| <code>pthread_cleanup_pop(3T)</code> | pop a thread cancellation cleanup handler |
| <code>pthread_cleanup_push(3T)</code> | push a thread cancellation cleanup handler |
| <code>pthread_cond_broadcast(3T)</code> | See <code>pthread_cond_signal(3T)</code> |
| <code>pthread_cond_destroy(3T)</code> | See <code>pthread_cond_init(3T)</code> |
| <code>pthread_cond_init(3T)</code> | initialize or destroy condition variables |
| <code>pthread_cond_signal(3T)</code> | signal or broadcast a condition |
| <code>pthread_cond_timedwait(3T)</code> | See <code>pthread_cond_wait(3T)</code> |

| | |
|---|--|
| <code>pthread_cond_wait(3T)</code> | wait on a condition |
| <code>pthread_condattr_destroy(3T)</code> | See <code>pthread_condattr_init(3T)</code> |
| <code>pthread_condattr_getpshared(3T)</code> | get or set the process-shared condition variable attributes |
| <code>pthread_condattr_init(3T)</code> | initialize or destroy condition variable attributes object |
| <code>pthread_condattr_setpshared(3T)</code> | See <code>pthread_condattr_getpshared(3T)</code> |
| <code>pthread_create(3T)</code> | create a thread |
| <code>pthread_detach(3T)</code> | detach a thread |
| <code>pthread_equal(3T)</code> | compare thread IDs |
| <code>pthread_exit(3T)</code> | terminate calling thread |
| <code>pthread_getconcurrency(3T)</code> | get or set level of concurrency |
| <code>pthread_getschedparam(3T)</code> | access dynamic thread scheduling parameters |
| <code>pthread_getspecific(3T)</code> | manage thread-specific data |
| <code>pthread_join(3T)</code> | wait for thread termination |
| <code>pthread_key_create(3T)</code> | create thread-specific data key |
| <code>pthread_key_delete(3T)</code> | delete thread-specific data key |
| <code>pthread_kill(3T)</code> | send a signal to a thread |
| <code>pthread_mutex_destroy(3T)</code> | See <code>pthread_mutex_init(3T)</code> |
| <code>pthread_mutex_getprioceiling(3T)</code> | change the priority ceiling of a mutex |
| <code>pthread_mutex_init(3T)</code> | initialize or destroy a mutex |
| <code>pthread_mutex_lock(3T)</code> | lock or unlock a mutex |

| | |
|---|---|
| <code>pthread_mutex_setprioceiling(3T)</code> | See <code>pthread_mutex_getprioceiling(3T)</code> |
| <code>pthread_mutex_trylock(3T)</code> | See <code>pthread_mutex_lock(3T)</code> |
| <code>pthread_mutex_unlock(3T)</code> | See <code>pthread_mutex_lock(3T)</code> |
| <code>pthread_mutexattr_destroy(3T)</code> | See <code>pthread_mutexattr_init(3T)</code> |
| <code>pthread_mutexattr_getprioceiling(3T)</code> | get and set prioceiling attribute of mutex attribute object |
| <code>pthread_mutexattr_getprotocol(3T)</code> | get and set protocol attribute of mutex attribute object |
| <code>pthread_mutexattr_getpshared(3T)</code> | get and set process-shared attribute |
| <code>pthread_mutexattr_gettype(3T)</code> | get or set a mutex type |
| <code>pthread_mutexattr_init(3T)</code> | initialize and destroy mutex attributes object |
| <code>pthread_mutexattr_setprioceiling(3T)</code> | See <code>pthread_mutexattr_getprioceiling(3T)</code> |
| <code>pthread_mutexattr_setprotocol(3T)</code> | See <code>pthread_mutexattr_getprotocol(3T)</code> |
| <code>pthread_mutexattr_setpshared(3T)</code> | See <code>pthread_mutexattr_getpshared(3T)</code> |
| <code>pthread_mutexattr_settype(3T)</code> | See <code>pthread_mutexattr_gettype(3T)</code> |
| <code>pthread_once(3T)</code> | initialize dynamic package |
| <code>pthread_rwlock_destroy(3T)</code> | See <code>pthread_rwlock_init(3T)</code> |
| <code>pthread_rwlock_init(3T)</code> | initialize or destroy a read-write lock object |

| | |
|--|--|
| <code>pthread_rwlock_rdlock(3T)</code> | lock or attempt to lock a read-write lock object for reading |
| <code>pthread_rwlock_tryrdlock(3T)</code> | See <code>pthread_rwlock_rdlock(3T)</code> |
| <code>pthread_rwlock_trywrlock(3T)</code> | See <code>pthread_rwlock_wrlock(3T)</code> |
| <code>pthread_rwlock_unlock(3T)</code> | unlock a read-write lock object |
| <code>pthread_rwlock_wrlock(3T)</code> | lock or attempt to lock a read-write lock object for writing |
| <code>pthread_rwlockattr_destroy(3T)</code> | See <code>pthread_rwlockattr_init(3T)</code> |
| <code>pthread_rwlockattr_getpshared(3T)</code> | get or set process-shared attribute of read-write lock attributes object |
| <code>pthread_rwlockattr_init(3T)</code> | initialize or destroy read-write lock attributes object |
| <code>pthread_rwlockattr_setpshared(3T)</code> | See <code>pthread_rwlockattr_getpshared(3T)</code> |
| <code>pthread_self(3T)</code> | get calling thread's ID |
| <code>pthread_setcancelstate(3T)</code> | enable or disable cancellation |
| <code>pthread_setcanceltype(3T)</code> | set the cancellation type of a thread |
| <code>pthread_setconcurrency(3T)</code> | See <code>pthread_getconcurrency(3T)</code> |
| <code>pthread_setschedparam(3T)</code> | See <code>pthread_getschedparam(3T)</code> |
| <code>pthread_setspecific(3T)</code> | See <code>pthread_getspecific(3T)</code> |

| | |
|-------------------------------|---|
| pthread_sigmask(3T) | change or examine calling thread's signal mask |
| pthread_testcancel(3T) | create cancellation point in the calling thread |
| pthreads(3T) | See threads(3T) |
| ptsname(3C) | get name of the slave pseudo-terminal device |
| publickey(3N) | See getpublickey(3N) |
| putc(3S) | See fputc(3S) |
| putc_unlocked(3S) | See fputc(3S) |
| putchar(3S) | See fputc(3S) |
| putchar_unlocked(3S) | See fputc(3S) |
| putenv(3C) | change or add value to environment |
| putmntent(3C) | See getmntent(3C) |
| putp(3X) | See curs_terminfo(3X) |
| putp(3XC) | apply padding information and output string |
| putpwent(3C) | write password file entry |
| puts(3S) | put a string on a stream |
| putspent(3C) | write shadow password file entry |
| pututline(3C) | See getutent(3C) |
| pututxline(3C) | See getutxent(3C) |
| putw(3S) | See fputc(3S) |
| putwc(3S) | See fputwc(3S) |
| putwchar(3S) | See fputwc(3S) |

| | |
|--------------------------------------|---|
| <code>putwin(3X)</code> | See <code> curs_util(3X)</code> |
| <code>putwin(3XC)</code> | See <code> getwin(3XC)</code> |
| <code>putws(3S)</code> | convert a string of Process Code characters to EUC characters |
| <code>qeconvert(3)</code> | See <code> econvert(3)</code> |
| <code>qfconvert(3)</code> | See <code> econvert(3)</code> |
| <code>qgconvert(3)</code> | See <code> econvert(3)</code> |
| <code>qiflush(3X)</code> | See <code> curs_inopts(3X)</code> |
| <code>qiflush(3XC)</code> | See <code> noqiflush(3XC)</code> |
| <code>qsort(3C)</code> | quick sort |
| <code>quadruple_to_decimal(3)</code> | See <code> floating_to_decimal(3)</code> |
| <code>rac_drop(3N)</code> | See <code> rpc_rac(3N)</code> |
| <code>rac_poll(3N)</code> | See <code> rpc_rac(3N)</code> |
| <code>rac_recv(3N)</code> | See <code> rpc_rac(3N)</code> |
| <code>rac_send(3N)</code> | See <code> rpc_rac(3N)</code> |
| <code>raise(3C)</code> | send signal to program |
| <code>rand(3B)</code> | simple random number generator |
| <code>rand(3C)</code> | simple random-number generator |
| <code>rand_r(3C)</code> | See <code> rand(3C)</code> |
| <code>random(3C)</code> | pseudorandom number functions |
| <code>raw(3X)</code> | See <code> curs_inopts(3X)</code> |
| <code>raw(3XC)</code> | See <code> cbreak(3XC)</code> |

| | |
|-----------------------|---|
| rcmd(3N) | routines for returning a stream to a remote command |
| re_comp(3C) | compile and execute regular expressions |
| re_exec(3C) | See re_comp(3C) |
| read_vtoc(3X) | read and write a disk's VTOC |
| readdir(3B) | read a directory entry |
| readdir(3C) | read directory |
| readdir_r(3C) | See readdir(3C) |
| realloc(3C) | See malloc(3C) |
| realloc(3X) | See malloc(3X) |
| realloc(3t) | See mtmalloc(3t) |
| realpath(3C) | resolve pathname |
| reboot(3C) | reboot system or halt processor |
| recv(3N) | receive a message from a socket |
| recv(3XN) | receive a message from a connected socket |
| recvfrom(3N) | See recv(3N) |
| recvfrom(3XN) | receive a message from a socket |
| recvmsg(3N) | See recv(3N) |
| recvmsg(3XN) | receive a message from a socket |
| redrawwin(3X) | See curs_refresh(3X) |
| redrawwin(3XC) | redraw screen or portion of screen |
| refresh(3X) | See curs_refresh(3X) |
| refresh(3XC) | See doupdate(3XC) |

| | |
|----------------------------------|---|
| <code>reg_ci_callback(3X)</code> | provide a component instrumentation with a transient program number |
| <code>regcmp(3C)</code> | compile and execute regular expression |
| <code>regcomp(3C)</code> | regular expression matching |
| <code>regerror(3C)</code> | See <code>regcomp(3C)</code> |
| <code>regex(3C)</code> | See <code>regcmp(3C)</code> |
| <code>regexec(3C)</code> | See <code>regcomp(3C)</code> |
| <code>regexpr(3G)</code> | regular expression compile and match routines |
| <code>regfree(3C)</code> | See <code>regcomp(3C)</code> |
| <code>registerrpc(3N)</code> | See <code>rpc_soc(3N)</code> |
| <code>remainder(3M)</code> | remainder function |
| <code>remove(3C)</code> | remove file |
| <code>remque(3C)</code> | See <code>insque(3C)</code> |
| <code>replace_panel(3X)</code> | See <code>panel_window(3X)</code> |
| <code>res_init(3N)</code> | See <code>resolver(3N)</code> |
| <code>res_mkquery(3N)</code> | See <code>resolver(3N)</code> |
| <code>res_mkupdate(3N)</code> | See <code>resolver(3N)</code> |
| <code>res_mkupdrec(3N)</code> | See <code>resolver(3N)</code> |
| <code>res_query(3N)</code> | See <code>resolver(3N)</code> |
| <code>res_search(3N)</code> | See <code>resolver(3N)</code> |
| <code>res_send(3N)</code> | See <code>resolver(3N)</code> |
| <code>res_update(3N)</code> | See <code>resolver(3N)</code> |
| <code>reset_prog_mode(3X)</code> | See <code>curs_kernel(3X)</code> |

| | |
|------------------------------------|--|
| <code>reset_prog_mode(3XC)</code> | See <code>def_prog_mode(3XC)</code> |
| <code>reset_shell_mode(3X)</code> | See <code>curs_kernel(3X)</code> |
| <code>reset_shell_mode(3XC)</code> | See <code>def_prog_mode(3XC)</code> |
| <code>resetty(3X)</code> | See <code>curs_kernel(3X)</code> |
| <code>resetty(3XC)</code> | restore/save terminal modes |
| <code>resolver(3N)</code> | resolver routines |
| <code>restartterm(3X)</code> | See <code>curs_terminfo(3X)</code> |
| <code>restartterm(3XC)</code> | See <code>del_curterm(3XC)</code> |
| <code>rewind(3S)</code> | reset file position indicator in a stream |
| <code>rewinddir(3C)</code> | reset position of directory stream to the beginning of a directory |
| <code>rexec(3N)</code> | return stream to a remote command |
| <code>rindex(3C)</code> | See <code>index(3C)</code> |
| <code>rint(3M)</code> | round-to-nearest integral value |
| <code>riporffline(3X)</code> | See <code>curs_kernel(3X)</code> |
| <code>riporffline(3XC)</code> | reserve screen line for dedicated purpose |
| <code>rmdirp(3G)</code> | See <code>mkdirp(3G)</code> |
| <code>rnusers(3N)</code> | See <code>rusers(3N)</code> |
| <code>rpc(3N)</code> | library routines for remote procedure calls |
| <code>rpc_broadcast(3N)</code> | See <code>rpc_clnt_calls(3N)</code> |
| <code>rpc_broadcast_exp(3N)</code> | See <code>rpc_clnt_calls(3N)</code> |
| <code>rpc_call(3N)</code> | See <code>rpc_clnt_calls(3N)</code> |

| | |
|---|---|
| <code>rpc_clnt_auth(3N)</code> | library routines for client side remote procedure call authentication |
| <code>rpc_clnt_calls(3N)</code> | library routines for client side calls |
| <code>rpc_clnt_create(3N)</code> | library routines for dealing with creation and manipulation of CLIENT handles |
| <code>rpc_control(3N)</code> | library routine for manipulating global RPC attributes for client and server applications |
| <code>rpc_createerr(3N)</code> | See <code>rpc_clnt_create(3N)</code> |
| <code>rpc_gss_get_error(3N)</code> | get error codes on failure |
| <code>rpc_gss_get_mech_info(3N)</code> | See <code>rpc_gss_get_mechanisms(3N)</code> |
| <code>rpc_gss_get_mechanisms(3N)</code> | get information on mechanisms and RPC version |
| <code>rpc_gss_get_principal_name(3N)</code> | Get principal names at server |
| <code>rpc_gss_get_versions(3N)</code> | See <code>rpc_gss_get_mechanisms(3N)</code> |
| <code>rpc_gss_getcred(3N)</code> | get credentials of client |
| <code>rpc_gss_is_installed(3N)</code> | See <code>rpc_gss_get_mechanisms(3N)</code> |
| <code>rpc_gss_max_data_length(3N)</code> | get maximum data length for transmission |
| <code>rpc_gss_mech_to_oid(3N)</code> | map mechanism, QOP strings to non-string values |
| <code>rpc_gss_qop_to_num(3N)</code> | See <code>rpc_gss_mech_to_oid(3N)</code> |
| <code>rpc_gss_seccreate(3N)</code> | create a security context using the |

| | |
|--|---|
| <code>rpc_gss_set_callback(3N)</code> | specify callback for context |
| <code>rpc_gss_set_defaults(3N)</code> | change service, QOP for a session |
| <code>rpc_gss_set_svc_name(3N)</code> | send a principal name to a server |
| <code>rpc_gss_svc_max_data_length(3N)</code> | See <code>rpc_gss_max_data_length(3N)</code> |
| <code>rpc_rac(3N)</code> | remote asynchronous calls |
| <code>rpc_reg(3N)</code> | See <code>rpc_svc_reg(3N)</code> |
| <code>rpc_soc(3N)</code> | obsolete library routines for RPC |
| <code>rpc_svc_calls(3N)</code> | library routines for RPC servers |
| <code>rpc_svc_create(3N)</code> | library routines for the creation of server handles |
| <code>rpc_svc_err(3N)</code> | library routines for server side remote procedure call errors |
| <code>rpc_svc_reg(3N)</code> | library routines for registering servers |
| <code>rpc_xdr(3N)</code> | XDR library routines for remote procedure calls |
| <code>rpcb_getaddr(3N)</code> | See <code>rpcbind(3N)</code> |
| <code>rpcb_getmaps(3N)</code> | See <code>rpcbind(3N)</code> |
| <code>rpcb_gettime(3N)</code> | See <code>rpcbind(3N)</code> |
| <code>rpcb_rmtcall(3N)</code> | See <code>rpcbind(3N)</code> |
| <code>rpcb_set(3N)</code> | See <code>rpcbind(3N)</code> |
| <code>rpcb_unset(3N)</code> | See <code>rpcbind(3N)</code> |
| <code>rpcbind(3N)</code> | library routines for RPC bind service |
| <code>rpcsec_gss(3N)</code> | security flavor incorporating |

| | |
|---------------------------|--|
| rresvport(3N) | See rcmd(3N) |
| rstat(3N) | get performance data from remote kernel |
| ruserok(3N) | See rcmd(3N) |
| rusers(3N) | return information about users on remote machines |
| rw_rdlock(3T) | See rwlock(3T) |
| rw_tryrdlock(3T) | See rwlock(3T) |
| rw_trywrlock(3T) | See rwlock(3T) |
| rw_unlock(3T) | See rwlock(3T) |
| rw_wrlock(3T) | See rwlock(3T) |
| rwall(3N) | write to specified remote machines |
| rwlock(3T) | multiple readers, single writer locks |
| rwlock_destroy(3T) | See rwlock(3T) |
| rwlock_init(3T) | See rwlock(3T) |
| savetty(3X) | See curs_kernel(3X) |
| savetty(3XC) | See resetty(3XC) |
| scalb(3M) | load exponent of a radix-independent floating-point number |
| scalbn(3M) | load exponent of a radix-independent floating-point number |
| scale_form(3X) | See form_win(3X) |
| scale_menu(3X) | See menu_win(3X) |
| scandir(3B) | scan a directory |

| | |
|-----------------------------------|--|
| scanf(3S) | convert formatted input |
| scanw(3X) | See curs_scanw(3X) |
| scanw(3XC) | See mvscanw(3XC) |
| sched_get_priority_max(3R) | get scheduling parameter limits |
| sched_get_priority_min(3R) | See sched_get_priority_max(3R) |
| sched_getparam(3R) | get scheduling parameters |
| sched_getscheduler(3R) | get scheduling policy |
| sched_rr_get_interval(3R) | get execution time limits |
| sched_setparam(3R) | set scheduling parameters |
| sched_setscheduler(3R) | set scheduling policy and scheduling parameters |
| sched_yield(3R) | yield processor |
| schedctl_exit(3X) | See schedctl_init(3X) |
| schedctl_init(3X) | preemption control |
| schedctl_lookup(3X) | See schedctl_init(3X) |
| schedctl_start(3X) | See schedctl_init(3X) |
| schedctl_stop(3X) | See schedctl_init(3X) |
| scr_dump(3X) | See curs_scr_dump(3X) |
| scr_dump(3XC) | write screen contents to/from a file |
| scr_init(3X) | See curs_scr_dump(3X) |
| scr_init(3XC) | See scr_dump(3XC) |
| scr_restore(3X) | See curs_scr_dump(3X) |
| scr_restore(3XC) | See scr_dump(3XC) |
| scr_set(3X) | See curs_scr_dump(3X) |

| | |
|-------------------------|--|
| scr_set(3XC) | See scr_dump(3XC) |
| scr1(3X) | See curs_scroll(3X) |
| scr1(3XC) | scroll a window |
| scroll(3X) | See curs_scroll(3X) |
| scroll(3XC) | See scr1(3XC) |
| scrollok(3X) | See curs_outopts(3X) |
| scrollok(3XC) | See clearok(3XC) |
| seconvert(3) | See econvert(3) |
| secure_rpc(3N) | library routines for secure remote procedure calls |
| seed48(3C) | See drand48(3C) |
| seekdir(3C) | set position of directory stream |
| select(3C) | synchronous I/O multiplexing |
| sem_close(3R) | close a named semaphore |
| sem_destroy(3R) | destroy an unnamed semaphore |
| sem_getvalue(3R) | get the value of a semaphore |
| sem_init(3R) | initialize an unnamed semaphore |
| sem_open(3R) | initialize/open a named semaphore |
| sem_post(3R) | increment the count of a semaphore |
| sem_trywait(3R) | See sem_wait(3R) |
| sem_unlink(3R) | remove a named semaphore |
| sem_wait(3R) | acquire or wait for a semaphore |
| sema_destroy(3T) | See semaphore(3T) |

| | |
|------------------------------|--|
| sema_init(3T) | See semaphore(3T) |
| sema_post(3T) | See semaphore(3T) |
| sema_trywait(3T) | See semaphore(3T) |
| sema_wait(3T) | See semaphore(3T) |
| semaphore(3T) | semaphores |
| send(3N) | send a message from a socket |
| send(3XN) | send a message on a socket |
| sendmsg(3N) | See send(3N) |
| sendmsg(3XN) | send a message on a socket using a message structure |
| sendto(3N) | See send(3N) |
| sendto(3XN) | send a message on a socket |
| set_current_field(3X) | See form_page(3X) |
| set_current_item(3X) | See menu_item_current(3X) |
| set_curterm(3X) | See curs_terminfo(3X) |
| set_curterm(3XC) | See del_curterm(3XC) |
| set_field_back(3X) | See form_field_attributes(3X) |
| set_field_buffer(3X) | See form_field_buffer(3X) |
| set_field_fore(3X) | See form_field_attributes(3X) |
| set_field_init(3X) | See form_hook(3X) |
| set_field_just(3X) | See form_field_just(3X) |
| set_field_opts(3X) | See form_field_opts(3X) |
| set_field_pad(3X) | See form_field_attributes(3X) |

| | |
|---------------------------------------|---|
| <code>set_field_status(3X)</code> | See <code>form_field_buffer(3X)</code> |
| <code>set_field_term(3X)</code> | See <code>form_hook(3X)</code> |
| <code>set_field_type(3X)</code> | See <code>form_field_validation(3X)</code> |
| <code>set_field_userptr(3X)</code> | See <code>form_field_userptr(3X)</code> |
| <code>set_fieldtype_arg(3X)</code> | See <code>form_fieldtype(3X)</code> |
| <code>set_fieldtype_choice(3X)</code> | See <code>form_fieldtype(3X)</code> |
| <code>set_form_fields(3X)</code> | See <code>form_field(3X)</code> |
| <code>set_form_init(3X)</code> | See <code>form_hook(3X)</code> |
| <code>set_form_opts(3X)</code> | See <code>form_opts(3X)</code> |
| <code>set_form_page(3X)</code> | See <code>form_page(3X)</code> |
| <code>set_form_sub(3X)</code> | See <code>form_win(3X)</code> |
| <code>set_form_term(3X)</code> | See <code>form_hook(3X)</code> |
| <code>set_form_userptr(3X)</code> | See <code>form_userptr(3X)</code> |
| <code>set_form_win(3X)</code> | See <code>form_win(3X)</code> |
| <code>set_item_init(3X)</code> | See <code>menu_hook(3X)</code> |
| <code>set_item_opts(3X)</code> | See <code>menu_item_opts(3X)</code> |
| <code>set_item_term(3X)</code> | See <code>menu_hook(3X)</code> |
| <code>set_item_userptr(3X)</code> | See <code>menu_item_userptr(3X)</code> |
| <code>set_item_value(3X)</code> | See <code>menu_item_value(3X)</code> |
| <code>set_max_field(3X)</code> | See <code>form_field_buffer(3X)</code> |
| <code>set_menu_back(3X)</code> | See <code>menu_attributes(3X)</code> |
| <code>set_menu_fore(3X)</code> | See <code>menu_attributes(3X)</code> |
| <code>set_menu_format(3X)</code> | See <code>menu_format(3X)</code> |
| <code>set_menu_grey(3X)</code> | See <code>menu_attributes(3X)</code> |

| | |
|------------------------------------|---|
| <code>set_menu_init(3X)</code> | See <code>menu_hook(3X)</code> |
| <code>set_menu_items(3X)</code> | See <code>menu_items(3X)</code> |
| <code>set_menu_mark(3X)</code> | See <code>menu_mark(3X)</code> |
| <code>set_menu_opts(3X)</code> | See <code>menu_opts(3X)</code> |
| <code>set_menu_pad(3X)</code> | See <code>menu_attributes(3X)</code> |
| <code>set_menu_pattern(3X)</code> | See <code>menu_pattern(3X)</code> |
| <code>set_menu_sub(3X)</code> | See <code>menu_win(3X)</code> |
| <code>set_menu_term(3X)</code> | See <code>menu_hook(3X)</code> |
| <code>set_menu_userptr(3X)</code> | See <code>menu_userptr(3X)</code> |
| <code>set_menu_win(3X)</code> | See <code>menu_win(3X)</code> |
| <code>set_new_page(3X)</code> | See <code>form_new_page(3X)</code> |
| <code>set_panel_userptr(3X)</code> | See <code>panel_userptr(3X)</code> |
| <code>set_term(3X)</code> | See <code>curs_initscr(3X)</code> |
| <code>set_term(3XC)</code> | switch between terminals |
| <code>set_top_row(3X)</code> | See <code>menu_item_current(3X)</code> |
| <code>setac(3)</code> | See <code>getacinfo(3)</code> |
| <code>setaclass(3)</code> | See <code>getaclassent(3)</code> |
| <code>setauevent(3)</code> | See <code>getauevent(3)</code> |
| <code>setauuser(3)</code> | See <code>getauusernam(3)</code> |
| <code>setbuf(3S)</code> | assign buffering to a stream |
| <code>setbuffer(3C)</code> | assign buffering to a stream |
| <code>setcat(3C)</code> | define default catalog |
| <code>setcchar(3XC)</code> | set a <code>cchar_t</code> type character from a wide character and rendition |
| <code>setgrent(3C)</code> | See <code>getgrnam(3C)</code> |

| | |
|-------------------------|-------------------------------------|
| sethostent(3N) | See gethostbyname(3N) |
| sethostent(3XN) | See endhostent(3XN) |
| sethostname(3C) | See gethostname(3C) |
| setjmp(3B) | non-local goto |
| setjmp(3C) | non-local goto |
| setkey(3C) | set encoding key |
| setlabel(3C) | define the label for |
| setlinebuf(3C) | See setbuffer(3C) |
| setlocale(3C) | modify and query a program's locale |
| setlogmask(3) | See syslog(3) |
| setnetconfig(3N) | See getnetconfig(3N) |
| setnetent(3N) | See getnetbyname(3N) |
| setnetent(3XN) | See endnetent(3XN) |
| setnetgrent(3N) | See getnetgrent(3N) |
| setnetpath(3N) | See getnetpath(3N) |
| setpriority(3C) | See getpriority(3C) |
| setprotoent(3N) | See getprotobyname(3N) |
| setprotoent(3XN) | See endprotoent(3XN) |
| setpwent(3C) | See getpwnam(3C) |
| setrpcent(3N) | See getrpcbyname(3N) |
| setscrreg(3X) | See curs_ouptops(3X) |
| setscrreg(3XC) | See clearok(3XC) |
| setservent(3N) | See getservbyname(3N) |
| setservent(3XN) | See endservent(3XN) |

| | |
|-------------------------|--|
| setsockopt(3N) | See getsockopt(3N) |
| setsockopt(3XN) | set the socket options |
| setspent(3C) | See getspnam(3C) |
| setstate(3C) | See random(3C) |
| setsyx(3X) | See curl_kernel(3X) |
| setterm(3X) | See curl_terminfo(3X) |
| setterm(3XC) | See del_curterm(3XC) |
| settimeofday(3B) | See gettimeofday(3B) |
| settimeofday(3C) | See gettimeofday(3C) |
| setupterm(3X) | See curl_terminfo(3X) |
| setupterm(3XC) | See del_curterm(3XC) |
| setusershell(3C) | See getusershell(3C) |
| setutent(3C) | See getutent(3C) |
| setutxent(3C) | See getutxent(3C) |
| setvbuf(3S) | See setbuf(3S) |
| sfconvert(3) | See econvert(3) |
| sgconvert(3) | See econvert(3) |
| shm_open(3R) | open a shared memory object |
| shm_unlink(3R) | remove a shared memory object |
| show_panel(3X) | See panel_show(3X) |
| shutdown(3N) | shut down part of a full-duplex connection |
| shutdown(3XN) | shut down socket send and receive operations |
| sig2str(3C) | See str2sig(3C) |
| sigaddset(3C) | See sigsetops(3C) |

| | |
|-------------------------|--|
| sigblock(3B) | block signals |
| sigdelset(3C) | See sigsetops(3C) |
| sigemptyset(3C) | See sigsetops(3C) |
| sigfillset(3C) | See sigsetops(3C) |
| sigfpe(3) | signal handling for specific SIGFPE codes |
| sighold(3C) | See signal(3C) |
| sigignore(3C) | See signal(3C) |
| siginterrupt(3B) | allow signals to interrupt functions |
| sigismember(3C) | See sigsetops(3C) |
| siglongjmp(3C) | See setjmp(3C) |
| sigmask(3B) | See sigblock(3B) |
| signal(3B) | simplified software signal facilities |
| signal(3C) | simplified signal management for application processes |
| significand(3M) | significand function |
| sigpause(3B) | See sigblock(3B) |
| sigpause(3C) | See signal(3C) |
| sigqueue(3R) | queue a signal to a process |
| sigrelse(3C) | See signal(3C) |
| sigset(3C) | See signal(3C) |
| sigsetjmp(3C) | See setjmp(3C) |
| sigsetmask(3B) | See sigblock(3B) |
| sigsetops(3C) | manipulate sets of signals |

| | |
|-----------------------------|---|
| sigstack(3B) | set and/or get signal stack context |
| sigstack(3C) | set and/or get alternate signal stack context |
| sigtimedwait(3R) | See sigwaitinfo(3R) |
| sigvec(3B) | software signal facilities |
| sigwaitinfo(3R) | wait for queued signals |
| sin(3M) | sine function |
| single_to_decimal(3) | See floating_to_decimal(3) |
| sinh(3M) | hyperbolic sine function |
| sleep(3B) | suspend execution for interval |
| sleep(3C) | suspend execution for an interval of time |
| slk_attr_off(3XC) | See slk_attroff(3XC) |
| slk_attr_on(3XC) | See slk_attroff(3XC) |
| slk_attr_set(3XC) | See slk_attroff(3XC) |
| slk_attroff(3X) | See curs_slk(3X) |
| slk_attroff(3XC) | manipulate soft labels |
| slk_attron(3X) | See curs_slk(3X) |
| slk_attron(3XC) | See slk_attroff(3XC) |
| slk_attrset(3X) | See curs_slk(3X) |
| slk_attrset(3XC) | See slk_attroff(3XC) |
| slk_clear(3X) | See curs_slk(3X) |
| slk_clear(3XC) | See slk_attroff(3XC) |
| slk_color(3XC) | See slk_attroff(3XC) |

| | |
|-----------------------------|---|
| slk_init(3X) | See curs_slk(3X) |
| slk_init(3XC) | See slk_attroff(3XC) |
| slk_label(3X) | See curs_slk(3X) |
| slk_label(3XC) | See slk_attroff(3XC) |
| slk_noutrefresh(3X) | See curs_slk(3X) |
| slk_noutrefresh(3XC) | See slk_attroff(3XC) |
| slk_refresh(3X) | See curs_slk(3X) |
| slk_refresh(3XC) | See slk_attroff(3XC) |
| slk_restore(3X) | See curs_slk(3X) |
| slk_restore(3XC) | See slk_attroff(3XC) |
| slk_set(3X) | See curs_slk(3X) |
| slk_set(3XC) | See slk_attroff(3XC) |
| slk_touch(3X) | See curs_slk(3X) |
| slk_touch(3XC) | See slk_attroff(3XC) |
| slk_wset(3XC) | See slk_attroff(3XC) |
| snprintf(3S) | See printf(3S) |
| socket(3N) | create an endpoint for communication |
| socket(3XN) | create an endpoint for communication |
| socketpair(3N) | create a pair of connected sockets |
| socketpair(3XN) | create a pair of connected sockets |
| space(3) | See plot(3) |
| spray(3N) | scatter data in order to test the network |

| | |
|-------------------------|---|
| sprintf(3B) | See printf(3B) |
| sprintf(3S) | See printf(3S) |
| sqrt(3M) | square root function |
| srand(3B) | See rand(3B) |
| srand(3C) | See rand(3C) |
| srand48(3C) | See drand48(3C) |
| srandom(3C) | See random(3C) |
| sscanf(3S) | See scanf(3S) |
| ssignal(3C) | software signals |
| standend(3X) | See curs_attr(3X) |
| standend(3XC) | set/clear window attributes |
| standout(3X) | See curs_attr(3X) |
| standout(3XC) | See standend(3XC) |
| start_color(3X) | See curs_color(3X) |
| start_color(3XC) | See can_change_color(3XC) |
| stdio(3S) | standard buffered input/output package |
| step(3G) | See regexpr(3G) |
| store(3B) | See dbm(3B) |
| str(3G) | See strfind(3G) |
| str2sig(3C) | translation between signal name and signal number |
| strcadd(3G) | See strccpy(3G) |
| strcasecmp(3C) | See string(3C) |
| strcat(3C) | See string(3C) |

| | |
|-----------------------------|---|
| strccpy(3G) | copy strings, compressing or expanding escape codes |
| strchr(3C) | See string(3C) |
| strcmp(3C) | See string(3C) |
| strcoll(3C) | string collation |
| strcpy(3C) | See string(3C) |
| strcspn(3C) | See string(3C) |
| strdup(3C) | See string(3C) |
| streadd(3G) | See strccpy(3G) |
| strecpy(3G) | See strccpy(3G) |
| strerror(3C) | get error message string |
| strfind(3G) | string manipulations |
| strfmon(3C) | convert monetary value to string |
| strftime(3C) | convert date and time to string |
| string(3C) | string operations |
| string_to_decimal(3) | parse characters into decimal record |
| strlen(3C) | See string(3C) |
| strncasecmp(3C) | See string(3C) |
| strncat(3C) | See string(3C) |
| strncmp(3C) | See string(3C) |
| strncpy(3C) | See string(3C) |
| strpbrk(3C) | See string(3C) |
| strptime(3C) | date and time conversion |
| strrchr(3C) | See string(3C) |

| | |
|-------------------------------|--|
| strrspn(3G) | See strfind(3G) |
| strsignal(3C) | get name of signal |
| strspn(3C) | See string(3C) |
| strstr(3C) | See string(3C) |
| strtod(3C) | convert string to double-precision number |
| strtok(3C) | See string(3C) |
| strtok_r(3C) | See string(3C) |
| strtol(3C) | string conversion routines |
| strtoll(3C) | See strtol(3C) |
| strtoul(3C) | convert string to unsigned long |
| strtoull(3C) | See strtoul(3C) |
| strtows(3C) | code conversion for Process Code and File Code |
| strtrns(3G) | See strfind(3G) |
| strxfrm(3C) | string transformation |
| subpad(3X) | See curs_pad(3X) |
| subpad(3XC) | See newpad(3XC) |
| subwin(3X) | See curs_window(3X) |
| subwin(3XC) | See derwin(3XC) |
| svc_auth_reg(3N) | See rpc_svc_reg(3N) |
| svc_control(3N) | See rpc_svc_create(3N) |
| svc_create(3N) | See rpc_svc_create(3N) |
| svc_destroy(3N) | See rpc_svc_create(3N) |
| svc_dg_create(3N) | See rpc_svc_create(3N) |
| svc_dg_enablecache(3N) | See rpc_svc_calls(3N) |

| | |
|------------------------------------|-------------------------------------|
| <code>svc_done(3N)</code> | See <code>rpc_svc_calls(3N)</code> |
| <code>svc_exit(3N)</code> | See <code>rpc_svc_calls(3N)</code> |
| <code>svc_fd_create(3N)</code> | See <code>rpc_svc_create(3N)</code> |
| <code>svc_fds(3N)</code> | See <code>rpc_soc(3N)</code> |
| <code>svc_fdset(3N)</code> | See <code>rpc_svc_calls(3N)</code> |
| <code>svc_freeargs(3N)</code> | See <code>rpc_svc_calls(3N)</code> |
| <code>svc_getargs(3N)</code> | See <code>rpc_svc_calls(3N)</code> |
| <code>svc_getcaller(3N)</code> | See <code>rpc_soc(3N)</code> |
| <code>svc_getreq(3N)</code> | See <code>rpc_soc(3N)</code> |
| <code>svc_getreq_common(3N)</code> | See <code>rpc_svc_calls(3N)</code> |
| <code>svc_getreq_poll(3N)</code> | See <code>rpc_svc_calls(3N)</code> |
| <code>svc_getreqset(3N)</code> | See <code>rpc_svc_calls(3N)</code> |
| <code>svc_getrpccaller(3N)</code> | See <code>rpc_svc_calls(3N)</code> |
| <code>svc_kerb_reg(3N)</code> | See <code>kerberos_rpc(3N)</code> |
| <code>svc_max_pollfd(3N)</code> | See <code>rpc_svc_calls(3N)</code> |
| <code>svc_pollfd(3N)</code> | See <code>rpc_svc_calls(3N)</code> |
| <code>svc_raw_create(3N)</code> | See <code>rpc_svc_create(3N)</code> |
| <code>svc_reg(3N)</code> | See <code>rpc_svc_reg(3N)</code> |
| <code>svc_register(3N)</code> | See <code>rpc_soc(3N)</code> |
| <code>svc_run(3N)</code> | See <code>rpc_svc_calls(3N)</code> |
| <code>svc_sendreply(3N)</code> | See <code>rpc_svc_calls(3N)</code> |
| <code>svc_tli_create(3N)</code> | See <code>rpc_svc_create(3N)</code> |
| <code>svc_tp_create(3N)</code> | See <code>rpc_svc_create(3N)</code> |
| <code>svc_unreg(3N)</code> | See <code>rpc_svc_reg(3N)</code> |
| <code>svc_unregister(3N)</code> | See <code>rpc_soc(3N)</code> |

| | |
|------------------------------------|---|
| svc_vc_create(3N) | See rpc_svc_create(3N) |
| svcerr_auth(3N) | See rpc_svc_err(3N) |
| svcerr_decode(3N) | See rpc_svc_err(3N) |
| svcerr_noproc(3N) | See rpc_svc_err(3N) |
| svcerr_noprogram(3N) | See rpc_svc_err(3N) |
| svcerr_progvers(3N) | See rpc_svc_err(3N) |
| svcerr_systemerr(3N) | See rpc_svc_err(3N) |
| svcerr_weakauth(3N) | See rpc_svc_err(3N) |
| svcfid_create(3N) | See rpc_soc(3N) |
| svccraw_create(3N) | See rpc_soc(3N) |
| svctcp_create(3N) | See rpc_soc(3N) |
| svcudp_bufcreate(3N) | See rpc_soc(3N) |
| svcudp_create(3N) | See rpc_soc(3N) |
| swab(3C) | swap bytes |
| swapcontext(3C) | See makecontext(3C) |
| swprintf(3S) | See fwprintf(3S) |
| swscanf(3S) | See fwscanf(3S) |
| sync_instruction_memory(3C) | make modified instructions executable |
| syncok(3X) | See curs_window(3X) |
| syncok(3XC) | synchronize window with its parents or children |
| sys_siglist(3B) | See psignal(3B) |
| syscall(3B) | indirect system call |
| sysconf(3C) | get configurable system variables |

| | |
|--------------------------------|---|
| <code>syslog(3)</code> | control system log |
| <code>system(3)</code> | return physical memory information |
| <code>system(3S)</code> | issue a shell command |
| <code>t_accept(3N)</code> | accept a connection request |
| <code>t_alloc(3N)</code> | allocate a library structure |
| <code>t_bind(3N)</code> | bind an address to a transport endpoint |
| <code>t_close(3N)</code> | close a transport endpoint |
| <code>t_connect(3N)</code> | establish a connection with another transport user |
| <code>t_errno(3N)</code> | XTI error return value |
| <code>t_error(3N)</code> | produce error message |
| <code>t_free(3N)</code> | free a library structure |
| <code>t_getinfo(3N)</code> | get protocol-specific service information |
| <code>t_getprotaddr(3N)</code> | get the protocol addresses |
| <code>t_getstate(3N)</code> | get the current state |
| <code>t_listen(3N)</code> | listen for a connection indication |
| <code>t_look(3N)</code> | look at the current event on a transport endpoint |
| <code>t_open(3N)</code> | establish a transport endpoint |
| <code>t_optmgmt(3N)</code> | manage options for a transport endpoint |
| <code>t_rcv(3N)</code> | receive data or expedited data sent over a connection |

| | |
|-------------------------------|--|
| <code>t_rcvconnect(3N)</code> | receive the confirmation from a connection request |
| <code>t_rcvdis(3N)</code> | retrieve information from disconnection |
| <code>t_rcvrel(3N)</code> | acknowledge receipt of an orderly release indication |
| <code>t_rcvreldata(3N)</code> | receive an orderly release indication or confirmation containing user data |
| <code>t_rcvudata(3N)</code> | receive a data unit |
| <code>t_rcvuderr(3N)</code> | receive a unit data error indication |
| <code>t_rcvv(3N)</code> | receive data or expedited data sent over a connection and put the data into one or more non-contiguous buffers |
| <code>t_rcvvudata(3N)</code> | receive a data unit into one or more noncontiguous buffers |
| <code>t_snd(3N)</code> | send data or expedited data over a connection |
| <code>t_snddis(3N)</code> | send user-initiated disconnection request |
| <code>t_sndrel(3N)</code> | initiate an orderly release |
| <code>t_sndreldata(3N)</code> | initiate or respond to an orderly release with user data |
| <code>t_sndudata(3N)</code> | send a data unit |
| <code>t_sndv(3N)</code> | send data or expedited data, from one or more non-contiguous buffers, on a connection |
| <code>t_sndvudata(3N)</code> | send a data unit from one or more noncontiguous buffers |

| | |
|----------------------------------|--|
| <code>t_strerror(3N)</code> | produce an error message string |
| <code>t_sync(3N)</code> | synchronize transport library |
| <code>t_sysconf(3N)</code> | get configurable XTI variables |
| <code>t_unbind(3N)</code> | disable a transport endpoint |
| <code>taddr2uaddr(3N)</code> | See <code>netdir(3N)</code> |
| <code>tan(3M)</code> | tangent function |
| <code>tanh(3M)</code> | hyperbolic tangent function |
| <code>tcdrain(3)</code> | wait for transmission of output |
| <code>tcflow(3)</code> | suspend or restart the transmission or reception of data |
| <code>tcflush(3)</code> | flush non-transmitted output data, non-read input data or both |
| <code>tcgetattr(3)</code> | get the parameters associated with the terminal |
| <code>tcgetpgrp(3)</code> | get foreground process group ID |
| <code>tcgetsid(3)</code> | get process group ID for session leader for controlling terminal |
| <code>tcsendbreak(3)</code> | send a "break" for a specific duration |
| <code>tcsetattr(3)</code> | set the parameters associated with the terminal |
| <code>tcsetpgrp(3)</code> | set foreground process group ID |
| <code>tcsetpgrp(3C)</code> | set foreground process group ID of terminal |
| <code>td_event_addset(3T)</code> | See <code>td_ta_event_addr(3T)</code> |
| <code>td_event_delset(3T)</code> | See <code>td_ta_event_addr(3T)</code> |

| | |
|--------------------------------------|---|
| <code>td_event_emptyset(3T)</code> | See <code>td_ta_event_addr(3T)</code> |
| <code>td_event_fillset(3T)</code> | See <code>td_ta_event_addr(3T)</code> |
| <code>td_eventisempty(3T)</code> | See <code>td_ta_event_addr(3T)</code> |
| <code>td_eventismember(3T)</code> | See <code>td_ta_event_addr(3T)</code> |
| <code>td_init(3T)</code> | performs initialization for libthread_db library of interfaces |
| <code>td_log(3T)</code> | placeholder for future logging functionality |
| <code>td_sync_get_info(3T)</code> | operations on a synchronization object in libthread_db |
| <code>td_sync_setstate(3T)</code> | See <code>td_sync_get_info(3T)</code> |
| <code>td_sync_waiters(3T)</code> | See <code>td_sync_get_info(3T)</code> |
| <code>td_ta_clear_event(3T)</code> | See <code>td_ta_event_addr(3T)</code> |
| <code>td_ta_delete(3T)</code> | See <code>td_ta_new(3T)</code> |
| <code>td_ta_enable_stats(3T)</code> | collect target process statistics for libthread_db |
| <code>td_ta_event_addr(3T)</code> | thread events in libthread_db |
| <code>td_ta_event_getmsg(3T)</code> | See <code>td_ta_event_addr(3T)</code> |
| <code>td_ta_get_nthreads(3T)</code> | gets the total number of threads in a process for libthread_db |
| <code>td_ta_get_ph(3T)</code> | See <code>td_ta_new(3T)</code> |
| <code>td_ta_get_stats(3T)</code> | See <code>td_ta_enable_stats(3T)</code> |
| <code>td_ta_map_addr2sync(3T)</code> | get a synchronization object handle from a synchronization object's address |
| <code>td_ta_map_id2thr(3T)</code> | convert a thread id or LWP id to a thread handle |

| | |
|---------------------------------------|--|
| <code>td_ta_map_lwp2thr(3T)</code> | See <code>td_ta_map_id2thr(3T)</code> |
| <code>td_ta_new(3T)</code> | allocate and deallocate process handles for <code>libthread_db</code> |
| <code>td_ta_reset_stats(3T)</code> | See <code>td_ta_enable_stats(3T)</code> |
| <code>td_ta_set_event(3T)</code> | See <code>td_ta_event_addr(3T)</code> |
| <code>td_ta_setconcurrency(3T)</code> | set concurrency level for target process |
| <code>td_ta_sync_iter(3T)</code> | iterator functions on process handles from <code>libthread_db</code> library of interfaces |
| <code>td_ta_thr_iter(3T)</code> | See <code>td_ta_sync_iter(3T)</code> |
| <code>td_ta_tsd_iter(3T)</code> | See <code>td_ta_sync_iter(3T)</code> |
| <code>td_thr_clear_event(3T)</code> | See <code>td_ta_event_addr(3T)</code> |
| <code>td_thr_dbresume(3T)</code> | See <code>td_thr_dbsuspend(3T)</code> |
| <code>td_thr_dbsuspend(3T)</code> | suspend and resume threads in <code>libthread_db</code> |
| <code>td_thr_event_enable(3T)</code> | See <code>td_ta_event_addr(3T)</code> |
| <code>td_thr_event_getmsg(3T)</code> | See <code>td_ta_event_addr(3T)</code> |
| <code>td_thr_get_info(3T)</code> | get thread information in <code>libthread_db</code> library of interfaces |
| <code>td_thr_getfpregs(3T)</code> | See <code>td_thr_getgregs(3T)</code> |
| <code>td_thr_getgregs(3T)</code> | reading and writing thread registers in <code>libthread_db</code> |
| <code>td_thr_getxregs(3T)</code> | See <code>td_thr_getgregs(3T)</code> |
| <code>td_thr_getxregsize(3T)</code> | See <code>td_thr_getgregs(3T)</code> |
| <code>td_thr_lockowner(3T)</code> | iterate over the set of locks owned by a thread |

| | |
|---------------------------------------|---|
| <code>td_thr_set_event(3T)</code> | See <code>td_ta_event_addr(3T)</code> |
| <code>td_thr_setfpregs(3T)</code> | See <code>td_thr_getgregs(3T)</code> |
| <code>td_thr_setgregs(3T)</code> | See <code>td_thr_getgregs(3T)</code> |
| <code>td_thr_setprio(3T)</code> | set the priority of a thread |
| <code>td_thr_setsigpending(3T)</code> | manage thread signals for <code>libthread_db</code> |
| <code>td_thr_setxregs(3T)</code> | See <code>td_thr_getgregs(3T)</code> |
| <code>td_thr_sigsetmask(3T)</code> | See <code>td_thr_setsigpending(3T)</code> |
| <code>td_thr_sleepinfo(3T)</code> | return the synchronization handle for the object on which a thread is blocked |
| <code>td_thr_tsd(3T)</code> | get a thread's thread-specific data for <code>libthread_db</code> library of interfaces |
| <code>td_thr_validate(3T)</code> | test a thread handle for validity |
| <code>tdelete(3C)</code> | See <code>tsearch(3C)</code> |
| <code>tell(3C)</code> | return a file offset for a file descriptor |
| <code>telldir(3C)</code> | current location of a named directory stream |
| <code>tempnam(3S)</code> | See <code>tmpnam(3S)</code> |
| <code>termattrs(3X)</code> | See <code>curls_termattrs(3X)</code> |
| <code>termattrs(3XC)</code> | return the video attributes supported by the terminal |
| <code>termios(3)</code> | general terminal interface |
| <code>termname(3X)</code> | See <code>curls_termattrs(3X)</code> |
| <code>termname(3XC)</code> | return the value of the environmental variable <code>TERM</code> |

| | |
|-------------------------------------|--|
| <code>textdomain(3C)</code> | See <code>gettext(3C)</code> |
| <code>tfind(3C)</code> | See <code>tsearch(3C)</code> |
| <code>tgetent(3X)</code> | See <code>curls_termcap(3X)</code> |
| <code>tgetent(3XC)</code> | emulate the termcap database |
| <code>tgetflag(3X)</code> | See <code>curls_termcap(3X)</code> |
| <code>tgetflag(3XC)</code> | See <code>tgetent(3XC)</code> |
| <code>tgetnum(3X)</code> | See <code>curls_termcap(3X)</code> |
| <code>tgetnum(3XC)</code> | See <code>tgetent(3XC)</code> |
| <code>tgetstr(3X)</code> | See <code>curls_termcap(3X)</code> |
| <code>tgetstr(3XC)</code> | See <code>tgetent(3XC)</code> |
| <code>tgoto(3X)</code> | See <code>curls_termcap(3X)</code> |
| <code>tgoto(3XC)</code> | See <code>tgetent(3XC)</code> |
| <code>thr_continue(3T)</code> | See <code>thr_suspend(3T)</code> |
| <code>thr_create(3T)</code> | create a thread |
| <code>thr_exit(3T)</code> | terminate the calling thread |
| <code>thr_getconcurrency(3T)</code> | get or set thread concurrency level |
| <code>thr_getprio(3T)</code> | access dynamic thread scheduling |
| <code>thr_getspecific(3T)</code> | See <code>thr_keycreate(3T)</code> |
| <code>thr_join(3T)</code> | wait for thread termination |
| <code>thr_keycreate(3T)</code> | thread-specific-data functions |
| <code>thr_kill(3T)</code> | send a signal to a thread |
| <code>thr_main(3T)</code> | identify the main thread |
| <code>thr_min_stack(3T)</code> | return the minimum-allowable size for a thread's stack |

| | |
|-------------------------------|---|
| thr_self(3T) | get calling thread's ID |
| thr_setconcurrency(3T) | See thr_getconcurrency(3T) |
| thr_setprio(3T) | See thr_getprio(3T) |
| thr_setspecific(3T) | See thr_keycreate(3T) |
| thr_sigsetmask(3T) | change or examine calling thread's signal mask |
| thr_stksegment(3T) | get thread stack bottom and stack size |
| thr_suspend(3T) | suspend or continue thread execution |
| thr_yield(3T) | yield to another thread |
| threads(3T) | concepts related to POSIX pthreads and Solaris threads and the libpthread and libthread libraries |
| tigetflag(3X) | See curls_terminfo(3X) |
| tigetflag(3XC) | return the value of a terminfo capability |
| tigetnum(3X) | See curls_terminfo(3X) |
| tigetnum(3XC) | See tigetflag(3XC) |
| tigetstr(3X) | See curls_terminfo(3X) |
| tigetstr(3XC) | See tigetflag(3XC) |
| timeout(3X) | See curls_inopts(3X) |
| timeout(3XC) | See notimeout(3XC) |
| timer_create(3R) | create a timer |
| timer_delete(3R) | delete a timer |
| timer_getoverrun(3R) | See timer_settime(3R) |

| | |
|--|--|
| <code>timer_gettime(3R)</code> | See <code>timer_settime(3R)</code> |
| <code>timer_settime(3R)</code> | per-process timers |
| <code>times(3B)</code> | get process times |
| <code>tmpfile(3S)</code> | create a temporary file |
| <code>tmpnam(3S)</code> | create a name for a temporary file |
| <code>tmpnam_r(3S)</code> | See <code>tmpnam(3S)</code> |
| <code>tnf_process_disable(3X)</code> | probe control internal interface |
| <code>tnf_process_enable(3X)</code> | See <code>tnf_process_disable(3X)</code> |
| <code>tnf_thread_disable(3X)</code> | See <code>tnf_process_disable(3X)</code> |
| <code>tnf_thread_enable(3X)</code> | See <code>tnf_process_disable(3X)</code> |
| <code>tnfctl_buffer_alloc(3X)</code> | allocate or deallocate a buffer for trace data |
| <code>tnfctl_buffer_dealloc(3X)</code> | See <code>tnfctl_buffer_alloc(3X)</code> |
| <code>tnfctl_check_libs(3X)</code> | See <code>tnfctl_indirect_open(3X)</code> |
| <code>tnfctl_close(3X)</code> | close a <code>tnfctl</code> handle |
| <code>tnfctl_continue(3X)</code> | See <code>tnfctl_pid_open(3X)</code> |
| <code>tnfctl_exec_open(3X)</code> | See <code>tnfctl_pid_open(3X)</code> |
| <code>tnfctl_filter_list_add(3X)</code> | See <code>tnfctl_trace_state_set(3X)</code> |
| <code>tnfctl_filter_list_delete(3X)</code> | See <code>tnfctl_trace_state_set(3X)</code> |
| <code>tnfctl_filter_list_get(3X)</code> | See <code>tnfctl_trace_state_set(3X)</code> |

| | |
|--|--|
| tnfctl_filter_state_set(3X) | See tnfctl_trace_state_set(3X) |
| tnfctl_indirect_open(3X) | control probes of another process where caller provides / proc functionality |
| tnfctl_internal_open(3X) | create handle for internal process probe control |
| tnfctl_kernel_open(3X) | create handle for kernel probe control |
| tnfctl_pid_open(3X) | interfaces for direct probe and process control for another process |
| tnfctl_probe_apply(3X) | iterate over probes |
| tnfctl_probe_apply_ids(3X) | See tnfctl_probe_apply(3X) |
| tnfctl_probe_connect(3X) | See tnfctl_probe_state_get(3X) |
| tnfctl_probe_disable(3X) | See tnfctl_probe_state_get(3X) |
| tnfctl_probe_disconnect_all(3X) | See tnfctl_probe_state_get(3X) |
| tnfctl_probe_enable(3X) | See tnfctl_probe_state_get(3X) |
| tnfctl_probe_state_get(3X) | interfaces to query and to change the state of a probe |
| tnfctl_probe_trace(3X) | See tnfctl_probe_state_get(3X) |
| tnfctl_probe_untrace(3X) | See tnfctl_probe_state_get(3X) |
| tnfctl_register_funcs(3X) | register callbacks for probe creation and destruction |

| | |
|---|--|
| <code>tnfctl_strerror(3X)</code> | map a <code>tnfctl</code> error code to a string |
| <code>tnfctl_trace_attrs_get(3X)</code> | get the trace attributes from a <code>tnfctl</code> handle |
| <code>tnfctl_trace_state_set(3X)</code> | control kernel tracing and process filtering |
| <code>toascii(3C)</code> | translate integer to a 7-bit ASCII character |
| <code>tolower(3C)</code> | transliterate upper-case characters to lower-case |
| <code>top_panel(3X)</code> | See <code>panel_top(3X)</code> |
| <code>top_row(3X)</code> | See <code>menu_item_current(3X)</code> |
| <code>touchline(3X)</code> | See <code>curltouch(3X)</code> |
| <code>touchline(3XC)</code> | See <code>is_linetouched(3XC)</code> |
| <code>touchlock(3X)</code> | See <code>maillock(3X)</code> |
| <code>touchwin(3X)</code> | See <code>curltouch(3X)</code> |
| <code>touchwin(3XC)</code> | See <code>is_linetouched(3XC)</code> |
| <code>toupper(3C)</code> | transliterate lower-case characters to upper-case |
| <code>towctrans(3C)</code> | wide-character mapping |
| <code>tolower(3C)</code> | transliterate upper-case wide-character code to lower-case |
| <code>toupper(3C)</code> | transliterate lower-case wide-character code to upper-case |
| <code>tparam(3X)</code> | See <code>curlterminfo(3X)</code> |
| <code>tparam(3XC)</code> | See <code>tigetflag(3XC)</code> |
| <code>tputs(3X)</code> | See <code>curlterminfo(3X)</code> |

| | |
|------------------------|--|
| tputs(3X) | See curs_terminfo(3X) |
| tputs(3XC) | See putp(3XC) |
| tracing(3X) | overview of tnf tracing system |
| truncate(3C) | set a file to a specified length |
| tsearch(3C) | manage binary search trees |
| ttyname(3C) | find pathname of a terminal |
| ttyname_r(3C) | See ttyname(3C) |
| ttyslot(3C) | find the slot in the utmp file of the current user |
| twalk(3C) | See tsearch(3C) |
| typeahead(3X) | See curs_inopts(3X) |
| typeahead(3XC) | check for type-ahead characters |
| tzset(3C) | See ctime(3C) |
| tzsetwall(3C) | See ctime(3C) |
| uaddr2taddr(3N) | See netdir(3N) |
| ualarm(3C) | schedule signal after interval in microseconds |
| ulckpddf(3C) | See lckpddf(3C) |
| ulltostr(3C) | See strtol(3C) |
| unctrl(3X) | See curs_util(3X) |
| unctrl(3XC) | convert character to printable form |
| unget_wch(3XC) | See ungetch(3XC) |
| ungetc(3S) | push byte back into input stream |
| ungetch(3X) | See curs_getch(3X) |

| | |
|--------------------------------|---|
| <code>ungetch(3XC)</code> | push character back onto the input queue |
| <code>ungetwc(3S)</code> | push wide-character code back into input stream |
| <code>ungetwch(3X)</code> | See <code>curs_getwch(3X)</code> |
| <code>unlockpt(3C)</code> | unlock a pseudo-terminal master/slave pair |
| <code>unordered(3C)</code> | See <code>isnan(3C)</code> |
| <code>unpost_form(3X)</code> | See <code>form_post(3X)</code> |
| <code>unpost_menu(3X)</code> | See <code>menu_post(3X)</code> |
| <code>untouchwin(3X)</code> | See <code>curs_touch(3X)</code> |
| <code>untouchwin(3XC)</code> | See <code>is_linetouched(3XC)</code> |
| <code>update_panels(3X)</code> | See <code>panel_update(3X)</code> |
| <code>updwtmp(3C)</code> | See <code>getutxent(3C)</code> |
| <code>updwtmpx(3C)</code> | See <code>getutxent(3C)</code> |
| <code>use_env(3X)</code> | See <code>curs_util(3X)</code> |
| <code>use_env(3XC)</code> | set values of lines and columns |
| <code>user2netname(3N)</code> | See <code>secure_rpc(3N)</code> |
| <code>usleep(3C)</code> | suspend execution for interval in microseconds |
| <code>utmpname(3C)</code> | See <code>getutent(3C)</code> |
| <code>utmpxname(3C)</code> | See <code>getutxent(3C)</code> |
| <code>valloc(3C)</code> | See <code>malloc(3C)</code> |
| <code>valloc(3X)</code> | See <code>watchmalloc(3X)</code> |
| <code>vfprintf(3B)</code> | See <code>printf(3B)</code> |
| <code>vfprintf(3S)</code> | See <code>vprintf(3S)</code> |

| | |
|-----------------------------------|--|
| vfwprintf(3S) | wide-character formatted output of a stdarg argument list |
| vid_attr(3XC) | See vidattr(3XC) |
| vid_puts(3XC) | See vidattr(3XC) |
| vidattr(3X) | See curs_terminfo(3X) |
| vidattr(3XC) | display string with video attributes |
| vidputs(3X) | See curs_terminfo(3X) |
| vidputs(3XC) | See vidattr(3XC) |
| vlfmt(3C) | display error message in standard format and pass to logging and monitoring services |
| vline(3XC) | See hline(3XC) |
| vline_set(3XC) | See hline_set(3XC) |
| volmgt_acquire(3X) | reserve removable media device |
| volmgt_check(3X) | have Volume Management check for media |
| volmgt_feature_enabled(3X) | check whether specific Volume Management features are enabled |
| volmgt_inuse(3X) | check whether or not Volume Management is managing a pathname |
| volmgt_ownspath(3X) | check Volume Management name space for path |
| volmgt_release(3X) | release removable media device reservation |
| volmgt_root(3X) | return the Volume Management root directory |

| | |
|---------------------------------|---|
| <code>volmgt_running(3X)</code> | return whether or not Volume Management is running |
| <code>volmgt_symdev(3X)</code> | See <code>volmgt_symname(3X)</code> |
| <code>volmgt_symname(3X)</code> | convert between Volume Management symbolic names, and the devices that correspond to them |
| <code>vpfmt(3C)</code> | display error message in standard format and pass to logging and monitoring services |
| <code>vprintf(3B)</code> | See <code>printf(3B)</code> |
| <code>vprintf(3S)</code> | print formatted output of a variable argument list |
| <code>vsnprintf(3S)</code> | See <code>vprintf(3S)</code> |
| <code>vsprintf(3B)</code> | See <code>printf(3B)</code> |
| <code>vsprintf(3S)</code> | See <code>vprintf(3S)</code> |
| <code>vswprintf(3S)</code> | See <code>vfwprintf(3S)</code> |
| <code>vsyslog(3)</code> | log message with a varargs argument list |
| <code>vw_printw(3XC)</code> | See <code>mvprintw(3XC)</code> |
| <code>vw_scanw(3XC)</code> | See <code>mvscanw(3XC)</code> |
| <code>vwprintf(3S)</code> | See <code>vfwprintf(3S)</code> |
| <code>vwprintw(3X)</code> | See <code> curs_printw(3X)</code> |
| <code>vwprintw(3XC)</code> | See <code>mvprintw(3XC)</code> |
| <code>vwscanw(3X)</code> | See <code> curs_scanw(3X)</code> |
| <code>vwscanw(3XC)</code> | See <code>mvscanw(3XC)</code> |
| <code>wadd_wch(3XC)</code> | See <code>add_wch(3XC)</code> |
| <code>wadd_wchnstr(3XC)</code> | See <code>add_wchnstr(3XC)</code> |

| | |
|-------------------------|---------------------------------------|
| wadd_wchstr(3XC) | See add_wchnstr(3XC) |
| waddch(3X) | See curs_addch(3X) |
| waddch(3XC) | See addch(3XC) |
| waddchnstr(3X) | See curs_addchstr(3X) |
| waddchnstr(3XC) | See addchstr(3XC) |
| waddchstr(3X) | See curs_addchstr(3X) |
| waddchstr(3XC) | See addchstr(3XC) |
| waddnstr(3X) | See curs_addstr(3X) |
| waddnstr(3XC) | See addnstr(3XC) |
| waddnwstr(3X) | See curs_addwstr(3X) |
| waddnwstr(3XC) | See addnwstr(3XC) |
| waddstr(3X) | See curs_addstr(3X) |
| waddstr(3XC) | See addnstr(3XC) |
| waddwch(3X) | See curs_addwch(3X) |
| waddwchnstr(3X) | See curs_addwchstr(3X) |
| waddwchstr(3X) | See curs_addwchstr(3X) |
| waddwstr(3X) | See curs_addwstr(3X) |
| waddwstr(3XC) | See addnwstr(3XC) |
| wadjcurspos(3X) | See curs_alecompat(3X) |
| wait(3B) | wait for process to terminate or stop |
| wait3(3B) | See wait(3B) |
| wait3(3C) | wait for process to terminate or stop |
| wait4(3B) | See wait(3B) |
| wait4(3C) | See wait3(3C) |

| | |
|-------------------------|-----------------------------|
| waitpid(3B) | See wait(3B) |
| watchmalloc(3X) | debugging memory allocator |
| watof(3C) | See wcstod(3C) |
| watoi(3C) | See wcstol(3C) |
| watol(3C) | See wcstol(3C) |
| watoll(3C) | See wcstol(3C) |
| wattr_get(3XC) | See attr_get(3XC) |
| wattr_off(3XC) | See attr_get(3XC) |
| wattr_on(3XC) | See attr_get(3XC) |
| wattr_set(3XC) | See attr_get(3XC) |
| wattroff(3X) | See curs_attr(3X) |
| wattroff(3XC) | See attroff(3XC) |
| wattron(3X) | See curs_attr(3X) |
| wattron(3XC) | See attroff(3XC) |
| wattrset(3X) | See curs_attr(3X) |
| wattrset(3XC) | See attroff(3XC) |
| wbkgd(3X) | See curs_bkgd(3X) |
| wbkgd(3XC) | See bkgd(3XC) |
| wbkgdset(3X) | See curs_bkgd(3X) |
| wbkgdset(3XC) | See bkgd(3XC) |
| wbkgdnd(3XC) | See bkgrnd(3XC) |
| wbkgdndset(3XC) | See bkgrnd(3XC) |
| wborder(3X) | See curs_border(3X) |
| wborder(3XC) | See border(3XC) |
| wborder_set(3XC) | See border_set(3XC) |

| | |
|------------------------------|--|
| <code>wchgat(3XC)</code> | See <code>chgat(3XC)</code> |
| <code>wclear(3X)</code> | See <code> curs_clear(3X)</code> |
| <code>wclear(3XC)</code> | See <code>clear(3XC)</code> |
| <code>wclrtoobot(3X)</code> | See <code> curs_clear(3X)</code> |
| <code>wclrtoobot(3XC)</code> | See <code>clrtoobot(3XC)</code> |
| <code>wclrtoeol(3X)</code> | See <code> curs_clear(3X)</code> |
| <code>wclrtoeol(3XC)</code> | See <code>clrtoeol(3XC)</code> |
| <code>wcolor_set(3XC)</code> | See <code>attr_get(3XC)</code> |
| <code>wcrtomb(3C)</code> | convert a wide-character code to a character (restartable) |
| <code>wscat(3C)</code> | See <code>wcstring(3C)</code> |
| <code>wchr(3C)</code> | See <code>wcstring(3C)</code> |
| <code>wscmp(3C)</code> | See <code>wcstring(3C)</code> |
| <code>wscoll(3C)</code> | wide character string comparison using collating information |
| <code>wscopy(3C)</code> | See <code>wcstring(3C)</code> |
| <code>wscspn(3C)</code> | See <code>wcstring(3C)</code> |
| <code>wcsetno(3C)</code> | See <code>cset(3C)</code> |
| <code>wcsftime(3C)</code> | convert date and time to wide character string |
| <code>wcslen(3C)</code> | See <code>wcstring(3C)</code> |
| <code>wcsncat(3C)</code> | See <code>wcstring(3C)</code> |
| <code>wcsncmp(3C)</code> | See <code>wcstring(3C)</code> |
| <code>wcsncpy(3C)</code> | See <code>wcstring(3C)</code> |
| <code>wcspbrk(3C)</code> | See <code>wcstring(3C)</code> |
| <code>wcsrchr(3C)</code> | See <code>wcstring(3C)</code> |

| | |
|------------------------------|---|
| <code>wcsrtombs(3C)</code> | convert a wide-character string to a character string (restartable) |
| <code>wcsspn(3C)</code> | See <code>wcstring(3C)</code> |
| <code>wcsstr(3C)</code> | find a wide-character substring |
| <code>wcstod(3C)</code> | convert wide character string to double-precision number |
| <code>wcstok(3C)</code> | See <code>wcstring(3C)</code> |
| <code>wcstol(3C)</code> | convert wide character string to long integer |
| <code>wcstombs(3C)</code> | convert a wide-character string to a character string |
| <code>wcstoul(3C)</code> | convert wide character string to unsigned long |
| <code>wcstring(3C)</code> | wide-character string operations |
| <code>wcswcs(3C)</code> | See <code>wcstring(3C)</code> |
| <code>wcswidth(3C)</code> | number of column positions of a wide-character string |
| <code>wcsxfrm(3C)</code> | wide character string transformation |
| <code>wctob(3C)</code> | wide-character to single-byte conversion |
| <code>wctomb(3C)</code> | convert a wide-character code to a character |
| <code>wctrans(3C)</code> | define character mapping |
| <code>wctype(3C)</code> | define character class |
| <code>wcursyncup(3X)</code> | See <code>curs_window(3X)</code> |
| <code>wcursyncup(3XC)</code> | See <code>syncok(3XC)</code> |
| <code>wcwidth(3C)</code> | number of column positions of a wide-character code |

| | |
|-------------------------------|-------------------------------------|
| <code>wdelch(3X)</code> | See <code> curs_delch(3X)</code> |
| <code>wdelch(3XC)</code> | See <code> delch(3XC)</code> |
| <code>wdeleteln(3X)</code> | See <code> curs_deleteln(3X)</code> |
| <code>wdeleteln(3XC)</code> | See <code> deleteln(3XC)</code> |
| <code>wecho_wchar(3XC)</code> | See <code> echo_wchar(3XC)</code> |
| <code>wechochar(3X)</code> | See <code> curs_addch(3X)</code> |
| <code>wechochar(3XC)</code> | See <code> echochar(3XC)</code> |
| <code>wechowchar(3X)</code> | See <code> curs_addwch(3X)</code> |
| <code>werase(3X)</code> | See <code> curs_clear(3X)</code> |
| <code>werase(3XC)</code> | See <code> clear(3XC)</code> |
| <code>wget_wch(3XC)</code> | See <code> get_wch(3XC)</code> |
| <code>wget_wstr(3XC)</code> | See <code> getn_wstr(3XC)</code> |
| <code>wgetbkgrnd(3XC)</code> | See <code> bkgrnd(3XC)</code> |
| <code>wgetch(3X)</code> | See <code> curs_getch(3X)</code> |
| <code>wgetch(3XC)</code> | See <code> getch(3XC)</code> |
| <code>wgetn_wstr(3XC)</code> | See <code> getn_wstr(3XC)</code> |
| <code>wgetnstr(3X)</code> | See <code> curs_getstr(3X)</code> |
| <code>wgetnstr(3XC)</code> | See <code> getnstr(3XC)</code> |
| <code>wgetnwstr(3X)</code> | See <code> curs_getwstr(3X)</code> |
| <code>wgetstr(3X)</code> | See <code> curs_getstr(3X)</code> |
| <code>wgetstr(3XC)</code> | See <code> getnstr(3XC)</code> |
| <code>wgetwch(3X)</code> | See <code> curs_getwch(3X)</code> |
| <code>wgetwstr(3X)</code> | See <code> curs_getwstr(3X)</code> |
| <code>whline(3X)</code> | See <code> curs_border(3X)</code> |
| <code>whline(3XC)</code> | See <code> hline(3XC)</code> |

| | |
|-------------------------------|-------------------------------------|
| <code>whline_set(3XC)</code> | See <code>hline_set(3XC)</code> |
| <code>win_wch(3XC)</code> | See <code>in_wch(3XC)</code> |
| <code>win_wchnstr(3XC)</code> | See <code>in_wchnstr(3XC)</code> |
| <code>win_wchstr(3XC)</code> | See <code>in_wchnstr(3XC)</code> |
| <code>winch(3X)</code> | See <code> curs_inch(3X)</code> |
| <code>winch(3XC)</code> | See <code> inch(3XC)</code> |
| <code>winchnstr(3X)</code> | See <code> curs_inchstr(3X)</code> |
| <code>winchnstr(3XC)</code> | See <code> inchnstr(3XC)</code> |
| <code>winchstr(3X)</code> | See <code> curs_inchstr(3X)</code> |
| <code>winchstr(3XC)</code> | See <code> inchnstr(3XC)</code> |
| <code>windex(3C)</code> | See <code> wcstring(3C)</code> |
| <code>winnstr(3X)</code> | See <code> curs_instr(3X)</code> |
| <code>winnstr(3XC)</code> | See <code> innstr(3XC)</code> |
| <code>winnwstr(3X)</code> | See <code> curs_inwstr(3X)</code> |
| <code>winnwstr(3XC)</code> | See <code> innwstr(3XC)</code> |
| <code>wins_nwstr(3XC)</code> | See <code> ins_nwstr(3XC)</code> |
| <code>wins_wch(3XC)</code> | See <code> ins_wch(3XC)</code> |
| <code>wins_wstr(3XC)</code> | See <code> ins_nwstr(3XC)</code> |
| <code>winsch(3X)</code> | See <code> curs_insch(3X)</code> |
| <code>winsch(3XC)</code> | See <code> insch(3XC)</code> |
| <code>winsdelln(3X)</code> | See <code> curs_deleteln(3X)</code> |
| <code>winsdelln(3XC)</code> | See <code> insdelln(3XC)</code> |
| <code>winsertln(3X)</code> | See <code> curs_deleteln(3X)</code> |
| <code>winsertln(3XC)</code> | See <code> insertln(3XC)</code> |
| <code>winsnstr(3X)</code> | See <code> curs_insstr(3X)</code> |

| | |
|------------------------|---|
| winsnstr(3XC) | See insnstr(3XC) |
| winsnwstr(3X) | See curs_inswstr(3X) |
| winsstr(3X) | See curs_insstr(3X) |
| winsstr(3XC) | See insnstr(3XC) |
| winstr(3X) | See curs_instr(3X) |
| winstr(3XC) | See innstr(3XC) |
| winswch(3X) | See curs_inswch(3X) |
| winswstr(3X) | See curs_inswstr(3X) |
| winwch(3X) | See curs_inwch(3X) |
| winwchnstr(3X) | See curs_inwchstr(3X) |
| winwchstr(3X) | See curs_inwchstr(3X) |
| winwstr(3X) | See curs_inwstr(3X) |
| winwstr(3XC) | See innwstr(3XC) |
| wmemchr(3C) | find a wide-character in memory |
| wmemcmp(3C) | compare wide-characters in memory |
| wmemcpy(3C) | copy wide-characters in memory |
| wmemmove(3C) | copy wide-characters in memory with overlapping areas |
| wmemset(3C) | set wide-characters in memory |
| wmove(3X) | See curs_move(3X) |
| wmove(3XC) | See move(3XC) |
| wmovenextch(3X) | See curs_alecompat(3X) |
| wmoveprevch(3X) | See curs_alecompat(3X) |

| | |
|--------------------------------|-----------------------------------|
| <code>wnoutrefresh(3X)</code> | See <code>curs_refresh(3X)</code> |
| <code>wnoutrefresh(3XC)</code> | See <code>douupdate(3XC)</code> |
| <code>wordexp(3C)</code> | perform word expansions |
| <code>wordfree(3C)</code> | See <code>wordexp(3C)</code> |
| <code>wprintf(3S)</code> | See <code>fwprintf(3S)</code> |
| <code>wprintw(3X)</code> | See <code>curs_printw(3X)</code> |
| <code>wprintw(3XC)</code> | See <code>mvprintw(3XC)</code> |
| <code>wredrawln(3X)</code> | See <code>curs_refresh(3X)</code> |
| <code>wredrawln(3XC)</code> | See <code>redrawwin(3XC)</code> |
| <code>wrefresh(3X)</code> | See <code>curs_refresh(3X)</code> |
| <code>wrefresh(3XC)</code> | See <code>douupdate(3XC)</code> |
| <code>wrindex(3C)</code> | See <code>wcstring(3C)</code> |
| <code>write_vtoc(3X)</code> | See <code>read_vtoc(3X)</code> |
| <code>wscanf(3S)</code> | See <code>fwscanf(3S)</code> |
| <code>wscanw(3X)</code> | See <code>curs_scanw(3X)</code> |
| <code>wscanw(3XC)</code> | See <code>mvscanw(3XC)</code> |
| <code>wscasecmp(3C)</code> | See <code>wstring(3C)</code> |
| <code>wscat(3C)</code> | See <code>wcstring(3C)</code> |
| <code>wchr(3C)</code> | See <code>wcstring(3C)</code> |
| <code>wscmp(3C)</code> | See <code>wcstring(3C)</code> |
| <code>wscol(3C)</code> | See <code>wstring(3C)</code> |
| <code>wscoll(3C)</code> | See <code>wcscoll(3C)</code> |
| <code>wscopy(3C)</code> | See <code>wcstring(3C)</code> |
| <code>wscr1(3X)</code> | See <code>curs_scroll(3X)</code> |
| <code>wscr1(3XC)</code> | See <code>schr1(3XC)</code> |

| | |
|------------------------|--------------------------------|
| wscspn(3C) | See wcstring(3C) |
| wsdup(3C) | See wstring(3C) |
| wsetscrreg(3X) | See curs_outopts(3X) |
| wsetscrreg(3XC) | See clearok(3XC) |
| wslen(3C) | See wcstring(3C) |
| wsncasecmp(3C) | See wstring(3C) |
| wsncat(3C) | See wcstring(3C) |
| wsncmp(3C) | See wcstring(3C) |
| wsncpy(3C) | See wcstring(3C) |
| wspbrk(3C) | See wcstring(3C) |
| wsprintf(3C) | formatted output conversion |
| wsrchr(3C) | See wcstring(3C) |
| wsscanf(3C) | formatted input conversion |
| wsspn(3C) | See wcstring(3C) |
| wstandend(3X) | See curs_attr(3X) |
| wstandend(3XC) | See standend(3XC) |
| wstandout(3X) | See curs_attr(3X) |
| wstandout(3XC) | See standend(3XC) |
| wstod(3C) | See wcstod(3C) |
| wstok(3C) | See wcstring(3C) |
| wstol(3C) | See wcstol(3C) |
| wstostr(3C) | See strtows(3C) |
| wstring(3C) | Process Code string operations |
| wsxfrm(3C) | See wcsxfrm(3C) |
| wsyncdown(3X) | See curs_window(3X) |

| | |
|-------------------------------------|---|
| <code>wsyncdown(3XC)</code> | See <code>syncok(3XC)</code> |
| <code>wsyncup(3X)</code> | See <code>curs_window(3X)</code> |
| <code>wsyncup(3XC)</code> | See <code>syncok(3XC)</code> |
| <code>wtimeout(3X)</code> | See <code>curs_inopts(3X)</code> |
| <code>wtimeout(3XC)</code> | See <code>notimeout(3XC)</code> |
| <code>wtouchln(3X)</code> | See <code>curs_touch(3X)</code> |
| <code>wtouchln(3XC)</code> | See <code>is_linetouched(3XC)</code> |
| <code>wunctrl(3XC)</code> | convert a wide character to printable form |
| <code>wvline(3X)</code> | See <code>curs_border(3X)</code> |
| <code>wvline(3XC)</code> | See <code>hline(3XC)</code> |
| <code>wvline_set(3XC)</code> | See <code>hline_set(3XC)</code> |
| <code>xdr(3N)</code> | library routines for external data representation |
| <code>xdr_accepted_reply(3N)</code> | See <code>rpc_xdr(3N)</code> |
| <code>xdr_admin(3N)</code> | library routines for external data representation |
| <code>xdr_array(3N)</code> | See <code>xdr_complex(3N)</code> |
| <code>xdr_authsys_parms(3N)</code> | See <code>rpc_xdr(3N)</code> |
| <code>xdr_authunix_parms(3N)</code> | See <code>rpc_soc(3N)</code> |
| <code>xdr_bool(3N)</code> | See <code>xdr_sim(3N)</code> |
| <code>xdr_bytes(3N)</code> | See <code>xdr_complex(3N)</code> |
| <code>xdr_callhdr(3N)</code> | See <code>rpc_xdr(3N)</code> |
| <code>xdr_callmsg(3N)</code> | See <code>rpc_xdr(3N)</code> |
| <code>xdr_char(3N)</code> | See <code>xdr_sim(3N)</code> |

| | |
|-------------------------------------|---|
| <code>xdr_complex(3N)</code> | library routines for external data representation |
| <code>xdr_control(3N)</code> | See <code>xdr_admin(3N)</code> |
| <code>xdr_create(3N)</code> | library routines for external data representation stream creation |
| <code>xdr_destroy(3N)</code> | See <code>xdr_create(3N)</code> |
| <code>xdr_double(3N)</code> | See <code>xdr_sim(3N)</code> |
| <code>xdr_enum(3N)</code> | See <code>xdr_sim(3N)</code> |
| <code>xdr_float(3N)</code> | See <code>xdr_sim(3N)</code> |
| <code>xdr_free(3N)</code> | See <code>xdr_sim(3N)</code> |
| <code>xdr_getpos(3N)</code> | See <code>xdr_admin(3N)</code> |
| <code>xdr_hyper(3N)</code> | See <code>xdr_sim(3N)</code> |
| <code>xdr_inline(3N)</code> | See <code>xdr_admin(3N)</code> |
| <code>xdr_int(3N)</code> | See <code>xdr_sim(3N)</code> |
| <code>xdr_long(3N)</code> | See <code>xdr_sim(3N)</code> |
| <code>xdr_longlong_t(3N)</code> | See <code>xdr_sim(3N)</code> |
| <code>xdr_opaque(3N)</code> | See <code>xdr_complex(3N)</code> |
| <code>xdr_opaque_auth(3N)</code> | See <code>rpc_xdr(3N)</code> |
| <code>xdr_pointer(3N)</code> | See <code>xdr_complex(3N)</code> |
| <code>xdr_quadruple(3N)</code> | See <code>xdr_sim(3N)</code> |
| <code>xdr_reference(3N)</code> | See <code>xdr_complex(3N)</code> |
| <code>xdr_rejected_reply(3N)</code> | See <code>rpc_xdr(3N)</code> |
| <code>xdr_replymsg(3N)</code> | See <code>rpc_xdr(3N)</code> |
| <code>xdr_setpos(3N)</code> | See <code>xdr_admin(3N)</code> |
| <code>xdr_short(3N)</code> | See <code>xdr_sim(3N)</code> |

| | |
|-------------------------------------|---|
| <code>xdr_sim(3N)</code> | library routines for external data representation |
| <code>xdr_sizeof(3N)</code> | See <code>xdr_admin(3N)</code> |
| <code>xdr_string(3N)</code> | See <code>xdr_complex(3N)</code> |
| <code>xdr_u_char(3N)</code> | See <code>xdr_sim(3N)</code> |
| <code>xdr_u_hyper(3N)</code> | See <code>xdr_sim(3N)</code> |
| <code>xdr_u_int(3N)</code> | See <code>xdr_sim(3N)</code> |
| <code>xdr_u_long(3N)</code> | See <code>xdr_sim(3N)</code> |
| <code>xdr_u_longlong_t(3N)</code> | See <code>xdr_sim(3N)</code> |
| <code>xdr_u_short(3N)</code> | See <code>xdr_sim(3N)</code> |
| <code>xdr_union(3N)</code> | See <code>xdr_complex(3N)</code> |
| <code>xdr_vector(3N)</code> | See <code>xdr_complex(3N)</code> |
| <code>xdr_void(3N)</code> | See <code>xdr_sim(3N)</code> |
| <code>xdr_wrapstring(3N)</code> | See <code>xdr_complex(3N)</code> |
| <code>xdrmem_create(3N)</code> | See <code>xdr_create(3N)</code> |
| <code>xdrrec_create(3N)</code> | See <code>xdr_create(3N)</code> |
| <code>xdrrec_endofrecord(3N)</code> | See <code>xdr_admin(3N)</code> |
| <code>xdrrec_eof(3N)</code> | See <code>xdr_admin(3N)</code> |
| <code>xdrrec_readbytes(3N)</code> | See <code>xdr_admin(3N)</code> |
| <code>xdrrec_skiprecord(3N)</code> | See <code>xdr_admin(3N)</code> |
| <code>xdrstdio_create(3N)</code> | See <code>xdr_create(3N)</code> |
| <code>xfn(3N)</code> | overview of the XFN interface |
| <code>xfn_attributes(3N)</code> | an overview of XFN attribute operations |

| | |
|--|---|
| <code>xfn_composite_names(3N)</code> | XFN composite syntax: an overview of the syntax for XFN composite name |
| <code>xfn_compound_names(3N)</code> | XFN compound syntax: an overview of XFN model for compound name parsing |
| <code>xfn_links(3N)</code> | XFN links: an overview of XFN links |
| <code>xfn_status_codes(3N)</code> | descriptions of XFN status codes |
| <code>xprt_register(3N)</code> | See <code>rpc_svc_reg(3N)</code> |
| <code>xprt_unregister(3N)</code> | See <code>rpc_svc_reg(3N)</code> |
| <code>y0(3M)</code> | Bessel functions of the second kind |
| <code>y1(3M)</code> | See <code>y0(3M)</code> |
| <code>yn(3M)</code> | See <code>y0(3M)</code> |
| <code>yp_all(3N)</code> | See <code>ypclnt(3N)</code> |
| <code>yp_bind(3N)</code> | See <code>ypclnt(3N)</code> |
| <code>yp_first(3N)</code> | See <code>ypclnt(3N)</code> |
| <code>yp_get_default_domain(3N)</code> | See <code>ypclnt(3N)</code> |
| <code>yp_master(3N)</code> | See <code>ypclnt(3N)</code> |
| <code>yp_match(3N)</code> | See <code>ypclnt(3N)</code> |
| <code>yp_next(3N)</code> | See <code>ypclnt(3N)</code> |
| <code>yp_order(3N)</code> | See <code>ypclnt(3N)</code> |
| <code>yp_unbind(3N)</code> | See <code>ypclnt(3N)</code> |
| <code>yp_update(3N)</code> | change NIS information |
| <code>ypclnt(3N)</code> | NIS Version 2 client interface |
| <code>yperr_string(3N)</code> | See <code>ypclnt(3N)</code> |

`ypprot_err(3N)`

See `ypclnt(3N)`

NAME a64l, l64a – convert between long integer and base-64 ASCII string

SYNOPSIS

```
#include <stdlib.h>

long a64l(const char * s);

char * l64a(long l);
```

DESCRIPTION These functions maintain numbers stored in base-64 ASCII characters that define a notation by which long integers can be represented by up to six characters. Each character represents a “digit” in a radix-64 notation.

The characters used to represent “digits” are as follows:

| Character | Digit |
|-----------|-------|
| . | 0 |
| / | 1 |
| 0-9 | 2-11 |
| A-Z | 12-37 |
| a-z | 38-63 |

The **a64l()** function takes a pointer to a null-terminated base-64 representation and returns a corresponding long value. If the string pointed to by *s* contains more than six characters, **a64l()** uses the first six.

The **a64l()** function scans the character string from left to right with the least significant digit on the left, decoding each character as a 6-bit radix-64 number.

The **l64a()** function takes a long argument and returns a pointer to the corresponding base-64 representation. If the argument is 0, **l64a()** returns a pointer to a null string.

The value returned by **l64a()** is a pointer into a static buffer, the contents of which are overwritten by each call. In the case of multithreaded applications, the return value is a pointer to thread specific data.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **attributes(5)**

| NAME | abort – terminate the process abnormally | | | | |
|----------------------|--|----------------|-----------------|----------|------|
| SYNOPSIS | #include <stdlib.h> void abort (void); | | | | |
| DESCRIPTION | The abort() function causes abnormal process termination to occur, unless the signal SIGABRT is being caught and the signal handler does not return. The abnormal termination processing includes at least the effect of fclose(3S) on all open streams and message catalogue descriptors, and the default actions defined for SIGABRT. The SIGABRT signal is sent to the calling process as if by means of the raise(3C) function with the argument SIGABRT. The status made available to wait(2) or waitpid(2) by abort will be that of a process terminated by the SIGABRT signal. abort will override blocking or ignoring the SIGABRT signal. | | | | |
| RETURN VALUES | The abort() function does not return. | | | | |
| ERRORS | No errors are defined. | | | | |
| USAGE | Catching the signal is intended to provide the application writer with a portable means to abort processing, free from possible interference from any implementation-provided library functions. If SIGABRT is neither caught nor ignored, and the current directory is writable, a core dump may be produced. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe | | | | |
| SEE ALSO | exit(2) , getrlimit(2) , kill(2) , wait(2) , waitpid(2) , fclose(3S) , raise(3C) , signal(3C) , attributes(5) | | | | |

| NAME | abs, labs, llabs – return absolute value of integer | | | | |
|--------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <stdlib.h> int abs(int val); long labs(long lval); long long llabs(long long llval);</pre> | | | | |
| DESCRIPTION | <p>The abs() function returns the absolute value of its <code>int</code> operand.</p> <p>The labs() function returns the absolute value of its <code>long</code> operand.</p> <p>The llabs() function returns the absolute value of its <code>long long</code> operand.</p> | | | | |
| USAGE | In 2's-complement representation, the absolute value of the largest magnitude negative integral value is undefined. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | attributes(5) | | | | |

| | | | | | |
|----------------------|--|--------------|----------------------------|--------------|---|
| NAME | accept – accept a connection on a socket | | | | |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lsocket -lnsl [<i>library</i> ...] #include <sys/types.h> #include <sys/socket.h></pre> | | | | |
| DESCRIPTION | <pre>int accept(int s, struct sockaddr *addr, socklen_t *addrlen);</pre> <p>The argument <i>s</i> is a socket that has been created with socket(3N) and bound to an address with bind(3N), and that is listening for connections after a call to listen(3N). The accept() function extracts the first connection on the queue of pending connections, creates a new socket with the properties of <i>s</i>, and allocates a new file descriptor, <i>ns</i>, for the socket. If no pending connections are present on the queue and the socket is not marked as non-blocking, accept() blocks the caller until a connection is present. If the socket is marked as non-blocking and no pending connections are present on the queue, accept() returns an error as described below. The accept() function uses the netconfig(4) file to determine the STREAMS device file name associated with <i>s</i>. This is the device on which the connect indication will be accepted. The accepted socket, <i>ns</i>, is used to read and write data to and from the socket that connected to <i>ns</i>; it is not used to accept more connections. The original socket (<i>s</i>) remains open for accepting further connections.</p> <p>The argument <i>addr</i> is a result parameter that is filled in with the address of the connecting entity as it is known to the communications layer. The exact format of the <i>addr</i> parameter is determined by the domain in which the communication occurs.</p> <p>The argument <i>addrlen</i> is a value-result parameter. Initially, it contains the amount of space pointed to by <i>addr</i>; on return it contains the length in bytes of the address returned.</p> <p>The accept() function is used with connection-based socket types, currently with SOCK_STREAM.</p> <p>It is possible to select(3C) or poll(2) a socket for the purpose of an accept() by selecting or polling it for a read. However, this will only indicate when a connect indication is pending; it is still necessary to call accept().</p> | | | | |
| RETURN VALUES | The accept() function returns -1 on error. If it succeeds, it returns a non-negative integer that is a descriptor for the accepted socket. | | | | |
| ERRORS | <p>accept() will fail if:</p> <table border="0"> <tr> <td style="padding-right: 20px;">EBADF</td> <td>The descriptor is invalid.</td> </tr> <tr> <td>EINTR</td> <td>The accept attempt was interrupted by the delivery of a signal.</td> </tr> </table> | EBADF | The descriptor is invalid. | EINTR | The accept attempt was interrupted by the delivery of a signal. |
| EBADF | The descriptor is invalid. | | | | |
| EINTR | The accept attempt was interrupted by the delivery of a signal. | | | | |

| | |
|--------------------|--|
| EMFILE | The per-process descriptor table is full. |
| ENODEV | The protocol family and type corresponding to <i>s</i> could not be found in the <code>netconfig</code> file. |
| ENOMEM | There was insufficient user memory available to complete the operation. |
| ENOSR | There were insufficient STREAMS resources available to complete the operation. |
| ENOTSOCK | The descriptor does not reference a socket. |
| EOPNOTSUPP | The referenced socket is not of type <code>SOCK_STREAM</code> . |
| EPROTO | A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized or the connection has already been released. |
| EWouldBlock | The socket is marked as non-blocking and no connections are present to be accepted. |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

`poll(2)`, `bind(3N)`, `connect(3N)`, `listen(3N)`, `select(3C)`, `socket(3N)`, `netconfig(4)`, `attributes(5)`, `socket(5)`

| | |
|--------------------|--|
| NAME | accept – accept a new connection on a socket |
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lxnet [<i>library ...</i>] #include <sys/socket.h></pre> <pre>int accept(int <i>socket</i>, struct sockaddr *<i>address</i>, socklen_t *<i>address_len</i>);</pre> |
| DESCRIPTION | <p>The accept() function extracts the first connection on the queue of pending connections, creates a new socket with the same socket type protocol and address family as the specified socket, and allocates a new file descriptor for that socket.</p> <p>The function takes the following arguments:</p> <p>socket Specifies a socket that was created with socket(3XN), has been bound to an address with bind(3XN), and has issued a successful call to listen(3XN).</p> <p>address Either a null pointer, or a pointer to a <code>sockaddr</code> structure where the address of the connecting socket will be returned.</p> <p>address_len Points to a <code>socklen_t</code> which on input specifies the length of the supplied <code>sockaddr</code> structure, and on output specifies the length of the stored address.</p> <p>If <i>address</i> is not a null pointer, the address of the peer for the accepted connection is stored in the <code>sockaddr</code> structure pointed to by <i>address</i>, and the length of this address is stored in the object pointed to by <i>address_len</i>.</p> <p>If the actual length of the address is greater than the length of the supplied <code>sockaddr</code> structure, the stored address will be truncated.</p> <p>If the protocol permits connections by unbound clients, and the peer is not bound, then the value stored in the object pointed to by <i>address</i> is unspecified.</p> <p>If the listen queue is empty of connection requests and <code>O_NONBLOCK</code> is not set on the file descriptor for the socket, accept() will block until a connection is present. If the listen(3XN) queue is empty of connection requests and <code>O_NONBLOCK</code> is set on the file descriptor for the socket, accept() will fail and set <code>errno</code> to <code>EAGAIN</code> or <code>EWOULDBLOCK</code>.</p> <p>The accepted socket cannot itself accept more connections. The original socket remains open and can accept more connections.</p> |
| USAGE | When a connection is available, select(3C) will indicate that the file descriptor for the socket is ready for reading. |

| | |
|----------------------|--|
| RETURN VALUES | Upon successful completion, accept() returns the nonnegative file descriptor of the accepted socket. Otherwise, -1 is returned and errno is set to indicate the error. |
| ERRORS | <p>The accept() function will fail if:</p> <p>EAGAIN EWOULDBLOCK O_NONBLOCK is set for the socket file descriptor and no connections are present to be accepted.</p> <p>EBADF The <i>socket</i> argument is not a valid file descriptor.</p> <p>ECONNABORTED A connection has been aborted.</p> <p>EFAULT The <i>address</i> or <i>address_len</i> parameter can not be accessed or written.</p> <p>EINTR The accept() function was interrupted by a signal that was caught before a valid connection arrived.</p> <p>EINVAL The <i>socket</i> is not accepting connections.</p> <p>EMFILE OPEN_MAX file descriptors are currently open in the calling process.</p> <p>ENFILE The maximum number of file descriptors in the system are already open.</p> <p>ENOTSOCK The <i>socket</i> argument does not refer to a socket.</p> <p>EOPNOTSUPP The socket type of the specified socket does not support accepting connections.</p> <p>The accept() function may fail if:</p> <p>ENOBUFS No buffer space is available.</p> <p>ENOMEM There was insufficient memory available to complete the operation.</p> <p>ENOSR There was insufficient STREAMS resources available to complete the operation.</p> <p>EPROTO A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized.</p> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

accept(3XN)

X/Open Networking Services Library Functions

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

bind(3XN), connect(3XN), listen(3XN), socket(3XN), attributes(5)

| | | | |
|----------------------|--|-----------|--|
| NAME | aclcheck – check the validity of an ACL | | |
| SYNOPSIS | <pre>#include <sys/acl.h> int aclcheck(aclent_t *aclbuf, int nentries, int *which);</pre> | | |
| DESCRIPTION | <p>aclcheck() checks the validity of an ACL pointed to by <i>aclbuf</i>. <i>nentries</i> is the number of entries contained in the buffer. <i>which</i> returns the index of the first entry that is invalid.</p> <p>The function verifies that an ACL pointed to by <i>aclbuf</i> is valid according to the following rules:</p> <ul style="list-style-type: none"> ■ There must be exactly one GROUP_OBJ ACL entry. ■ There must be exactly one USER_OBJ ACL entry. ■ There must be exactly one OTHER_OBJ ACL entry. ■ If there are any GROUP ACL entries, then the group ID in each group ACL entry must be unique. ■ If there are any USER ACL entries, then the user ID in each user ACL entry must be unique. ■ If there are any GROUP or USER ACL entries, then there must be exactly one CLASS_OBJ (ACL mask) entry. ■ If there are any default ACL entries, then the following apply: <ul style="list-style-type: none"> ■ There must be exactly one default GROUP_OBJ ACL entry. ■ There must be exactly one default OTHER_OBJ ACL entry. ■ There must be exactly one default USER_OBJ ACL entry. ■ If there are any DEF_GROUP entries, then the group ID in each DEF_GROUP ACL entry must be unique. ■ If there are any DEF_USER entries, then the user ID in each DEF_USER ACL entry must be unique. ■ If there are any DEF_GROUP or DEF_USER entries, then there must be exactly one DEF_CLASS_OBJ (default ACL mask) entry. ■ If any of the above rules are violated, then the function fails with <i>errno</i> set to EINVAL. | | |
| RETURN VALUES | <p>If the ACL is valid, aclcheck() will return 0. Otherwise <i>errno</i> is set to EINVAL and return code is set to one of the following.</p> <table border="0" style="width: 100%;"> <tr> <td style="width: 50%; vertical-align: top;">GRP_ERROR</td> <td style="vertical-align: top;">There is more than one GROUP_OBJ or DEF_GROUP_OBJ ACL entry.</td> </tr> </table> | GRP_ERROR | There is more than one GROUP_OBJ or DEF_GROUP_OBJ ACL entry. |
| GRP_ERROR | There is more than one GROUP_OBJ or DEF_GROUP_OBJ ACL entry. | | |

| | |
|-----------------|--|
| USER_ERROR | There is more than one USER_OBJ or DEF_USER_OBJ ACL entry. |
| CLASS_ERROR | There is more than one CLASS_OBJ (ACL mask) or DEF_CLASS_OBJ (default ACL mask) entry. |
| OTHER_ERROR | There is more than one OTHER_OBJ or DEF_OTHER_OBJ ACL entry. |
| DUPLICATE_ERROR | Duplicate entries of USER, GROUP, DEF_USER, or DEF_GROUP. |
| ENTRY_ERROR | The entry type is invalid. |
| MISS_ERROR | Missing an entry. which returns -1 in this case. |
| MEM_ERROR | The system can't allocate any memory. which returns -1 in this case. |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Evolving |

SEE ALSO

acl(2), **aclsort(3)**

NAME | `aclsort` – sort an ACL

SYNOPSIS | `#include <sys/acl.h>`

int `aclsort`(int *nentries*, int *calclass*, `aclent_t *aclbufp`);

DESCRIPTION

aclbufp points to a buffer containing ACL entries. *nentries* specifies the number of ACL entries in the buffer. *calclass*, if non-zero, indicates that the `CLASS_OBJ` (ACL mask) permissions should be recalculated. The union of the permission bits associated with all ACL entries in the buffer other than `CLASS_OBJ`, `OTHER_OBJ`, and `USER_OBJ` is calculated. The result is copied to the permission bits associated with the `CLASS_OBJ` entry.

`aclsort()` sorts the contents of the ACL buffer as follows:

- Entries will be in the order `USER_OBJ`, `USER`, `GROUP_OBJ`, `GROUP`, `CLASS_OBJ` (ACL mask), `OTHER_OBJ`, `DEF_USER_OBJ`, `DEF_USER`, `DEF_GROUP_OBJ`, `DEF_GROUP`, `DEF_CLASS_OBJ` (default ACL mask), and `DEF_OTHER_OBJ`.
- Entries of type `USER`, `GROUP`, `DEF_USER`, and `DEF_GROUP` will be sorted in increasing order by id.

`aclsort()` will succeed if all of the following are true:

- There is exactly one entry each of type `USER_OBJ`, `GROUP_OBJ`, `CLASS_OBJ` (ACL mask), and `OTHER_OBJ`.
- There is exactly one entry each of type `DEF_USER_OBJ`, `DEF_GROUP_OBJ`, `DEF_CLASS_OBJ` (default ACL mask), and `DEF_OTHER_OBJ` if there are any default entries.
- Entries of type `USER`, `GROUP`, `DEF_USER`, or `DEF_GROUP` may not contain duplicate entries. A duplicate entry is one of the same type containing the same numeric id.

RETURN VALUES

Upon successful completion, the return value is 0. Otherwise, the return value is -1.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Evolving |

SEE ALSO

`acl(2)`, `aclcheck(3)`

NAME | `acltomode`, `aclfrommode` – convert an ACL to/from permission bits

SYNOPSIS | `#include <sys/types.h>`
| `#include <sys/acl.h>`

| `int acltomode(aclent_t * aclbufp, int nentries, mode_t * modep);`
| `int aclfrommode(aclent_t * aclbufp, int nentries, mode_t * modep);`

DESCRIPTION | `acltomode()` converts an ACL pointed to by `aclbufp` into the permission bits buffer pointed to by `modep`. If the `USER_OBJ` ACL entry, `GROUP_OBJ` ACL entry, or the `OTHER_OBJ` ACL entry cannot be found in the ACL buffer, then the function fails with `errno` set to `EINVAL`.

| The `USER_OBJ` ACL entry permission bits are copied to the file owner permission bits in the permission bits buffer. The `OTHER_OBJ` ACL entry permission bits are copied to the file other permission bits in the permission bits buffer. If there is a `CLASS_OBJ` (ACL mask) entry, then the `CLASS_OBJ` ACL entry permission bits are intersected (bitwise AND) with the `GROUP_OBJ` ACL entry permission bits and the result is copied to the file group permission bits in the permission bits buffer. Otherwise, the `GROUP_OWNER` ACL entry permission bits are copied to the file group permission bits in the permission bits buffer.

| `aclfrommode()` converts the permission bits pointed to by `modep` into an ACL pointed to by `aclbufp`. If the `USER_OBJ` ACL entry, `GROUP_OBJ` ACL entry, or the `OTHER_OBJ` ACL entry cannot be found in the ACL buffer, then the function fails with `errno` set to `EINVAL`.

| The file owner permission bits from the permission bits buffer are copied to the `USER_OBJ` ACL entry. The file other permission bits from the permission bits buffer are copied to the `OTHER_OBJ` ACL entry. The file group permissions bits from the permission bits buffer are copied to the `CLASS_OBJ` (ACL mask) entry, if available, and to the `GROUP_OBJ` ACL entry.

| `nentries` is the number of ACL entries in the buffer pointed to by `aclbufp`.

RETURN VALUES | Upon successful completion, the function returns 0. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

ATTRIBUTES | See `attributes` (5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Evolving |

SEE ALSO

ac1(2)

| | | | | | | | | | | | |
|--------------------|---|------|---|-------|--|-------|---|------|---|--------------|---|
| NAME | acltotext, aclfromtext – convert an internal representation to/from external representation | | | | | | | | | | |
| SYNOPSIS | <pre>#include <sys/acl.h> char * acltotext(aclent_t * <i>aclbufp</i>, int <i>aclcnt</i>); aclent_t * aclfromtext(char * <i>acltextp</i>, int * <i>aclcnt</i>);</pre> | | | | | | | | | | |
| DESCRIPTION | <p>acltotext() converts an internal ACL representation pointed to by <i>aclbufp</i> into an external ACL representation. The space for the external text string is obtained using <code>malloc(3C)</code>. The caller is responsible for freeing the space when it's done.</p> <p>aclfromtext() converts an external ACL representation pointed to by <i>acltextp</i> into an internal ACL representation. The space for the list of ACL entries is obtained using <code>malloc(3C)</code>. The caller is responsible for freeing the space when it's done. <i>aclcnt</i> is returned to indicate the number of acl entries found.</p> <p>An external ACL representation is defined as follows:</p> <pre><acl_entry>[, <acl_entry>] . . .</pre> <p>Each <acl_entry> contains one ACL entry. The external representation of an ACL entry contains two or three colon-separated fields. The first field contains the ACL entry tag type. The entry type keywords are defined as:</p> <table border="0"> <tr> <td style="padding-right: 20px;">user</td> <td>This ACL entry with no uid specified in the ACL entry id field specifies the access granted to the owner of the object. Otherwise, this ACL entry specifies the access granted to a specific user-name or user-id number.</td> </tr> <tr> <td>group</td> <td>This ACL entry with no gid specified in the ACL entry id field specifies the access granted to the owning group of the object. Otherwise, this ACL entry specifies the access granted to a specific group-name or group-id number.</td> </tr> <tr> <td>other</td> <td>This ACL entry specifies the access granted to any user or group that does not match any other ACL entry.</td> </tr> <tr> <td>mask</td> <td>This ACL entry specifies the maximum access granted to user or group entries.</td> </tr> <tr> <td>default:user</td> <td>This ACL entry with no uid specified in the ACL entry id field specifies the default access granted to the owner of the object. Otherwise, this ACL entry specifies the default access granted to a specific user-name or user-id number.</td> </tr> </table> | user | This ACL entry with no uid specified in the ACL entry id field specifies the access granted to the owner of the object. Otherwise, this ACL entry specifies the access granted to a specific user-name or user-id number. | group | This ACL entry with no gid specified in the ACL entry id field specifies the access granted to the owning group of the object. Otherwise, this ACL entry specifies the access granted to a specific group-name or group-id number. | other | This ACL entry specifies the access granted to any user or group that does not match any other ACL entry. | mask | This ACL entry specifies the maximum access granted to user or group entries. | default:user | This ACL entry with no uid specified in the ACL entry id field specifies the default access granted to the owner of the object. Otherwise, this ACL entry specifies the default access granted to a specific user-name or user-id number. |
| user | This ACL entry with no uid specified in the ACL entry id field specifies the access granted to the owner of the object. Otherwise, this ACL entry specifies the access granted to a specific user-name or user-id number. | | | | | | | | | | |
| group | This ACL entry with no gid specified in the ACL entry id field specifies the access granted to the owning group of the object. Otherwise, this ACL entry specifies the access granted to a specific group-name or group-id number. | | | | | | | | | | |
| other | This ACL entry specifies the access granted to any user or group that does not match any other ACL entry. | | | | | | | | | | |
| mask | This ACL entry specifies the maximum access granted to user or group entries. | | | | | | | | | | |
| default:user | This ACL entry with no uid specified in the ACL entry id field specifies the default access granted to the owner of the object. Otherwise, this ACL entry specifies the default access granted to a specific user-name or user-id number. | | | | | | | | | | |

`default:group` This ACL entry with no gid specified in the ACL entry id field specifies the default access granted to the owning group of the object. Otherwise, this ACL entry specifies the default access granted to a specific group-name or group-id number.

`default:other` This ACL entry specifies the default access for other entry.

`default:mask` This ACL entry specifies the default access for mask entry. The second field contains the ACL entry id. It is as follows:

`uid` This field specifies a user-name, or user-id if there is no user-name associated with the user-id number.

`gid` This field specifies a group-name, or group-id if there is no group-name associated with the group-id number.

`empty` It is used by the user and group ACL entry types. The third field contains the following symbolic discretionary access permissions:

`r` read permission

`w` write permission

`x` execute/search permission

`-` no access

RETURN VALUES

Upon successful completion, the function returns a pointer to a text string (`acltotext()`) or to a list of ACL entries (`aclfromtext()`). Otherwise, it returns `NULL`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---------------------|-----------------|
| Interface Stability | Evolving |

SEE ALSO

`acl(2)` , `malloc(3C)`

| NAME | acos – arc cosine function | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | cc [<i>flag ...</i>] <i>file ...</i> -lm [<i>library ...</i>] #include <math.h> double acos (double <i>x</i>); | | | | |
| DESCRIPTION | The acos() function computes the principal value of the arc cosine of <i>x</i> . The value of <i>x</i> should be in the range [-1,1]. | | | | |
| RETURN VALUES | Upon successful completion, acos() returns the arc cosine of <i>x</i> , in the range [0,pi] radians. If the value of <i>x</i> is not in the range [-1,1], and is not ±Inf or NaN, either 0.0 or NaN is returned and <code>errno</code> is set to EDOM. If <i>x</i> is NaN, NaN is returned. If <i>x</i> is ±Inf, either 0.0 is returned and <code>errno</code> is set to EDOM, or NaN is returned and <code>errno</code> may be set to EDOM. For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4. | | | | |
| ERRORS | The acos() function will fail if: EDOM The value <i>x</i> is not ±Inf or NaN and is not in the range [-1,1]. The acos() function may fail if: EDOM The value <i>x</i> is ±Inf. | | | | |
| USAGE | An application wishing to check for error situations should set <code>errno</code> to 0 before calling acos() . If <code>errno</code> is non-zero on return, or the value NaN is returned, an error has occurred. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | cos(3M) , isnan(3M) , matherr(3M) , attributes(5) , standards(5) | | | | |

| | |
|----------------------|---|
| NAME | acosh, asinh, atanh – inverse hyperbolic functions |
| SYNOPSIS | <pre>cc [flag ...] file ... -lm [library ...] #include <math.h> double acosh(double x); double asinh(double x); double atanh(double x);</pre> |
| DESCRIPTION | The acosh() , asinh() and atanh() functions compute the inverse hyperbolic cosine, sine, and tangent of their argument, respectively. |
| RETURN VALUES | <p>The acosh() , asinh() and atanh() functions return the inverse hyperbolic cosine, sine, and tangent of their argument, respectively.</p> <p>The acosh() function returns NaN and sets <code>errno</code> to EDOM when its argument is less than 1.0.</p> <p>The atanh() function returns NaN and sets <code>errno</code> to EDOM when its argument has absolute value greater than 1.0.</p> <p>The atanh() function returns $\pm\text{Inf}$ and sets <code>errno</code> to ERANGE when its argument is ± 1.0.</p> <p>If x is NaN, the asinh() , acosh() and atanh() functions return NaN.</p> <p>For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.</p> |
| ERRORS | <p>The acosh() function will fail if:</p> <p>EDOM The x argument is less than 1.0.</p> <p>The atanh() function will fail if:</p> <p>EDOM The x argument has an absolute value greater than 1.0.</p> <p>ERANGE The x argument has an absolute value equal to 1.0</p> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`cosh(3M)` , `matherr(3M)` , `sinh(3M)` , `tanh(3M)` , `attributes(5)` ,
`standards(5)`

| | | | | | | | | | |
|--------------------|---|------------------|--|-----------------|--|-----------------|---|-------------------|---|
| NAME | addch, mvaddch, mvwaddch, waddch – add a character (with rendition) to a window | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int addch(const chtype ch); int mvaddch(int y, int x, const chtype ch); int mvwaddch(WINDOW * win, int y, int x, const chtype ch); int waddch(WINDOW * win, const chtype ch);</pre> | | | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="padding-right: 10px;"><i>ch</i></td> <td>Is the character/attribute pair to be written to the window.</td> </tr> <tr> <td style="padding-right: 10px;"><i>y</i></td> <td>Is the y (row) coordinate of the character's position in the window.</td> </tr> <tr> <td style="padding-right: 10px;"><i>x</i></td> <td>Is the x (column) coordinate of the character's position in the window.</td> </tr> <tr> <td style="padding-right: 10px;"><i>win</i></td> <td>Is a pointer to the window in which the character is to be written.</td> </tr> </table> | <i>ch</i> | Is the character/attribute pair to be written to the window. | <i>y</i> | Is the y (row) coordinate of the character's position in the window. | <i>x</i> | Is the x (column) coordinate of the character's position in the window. | <i>win</i> | Is a pointer to the window in which the character is to be written. |
| <i>ch</i> | Is the character/attribute pair to be written to the window. | | | | | | | | |
| <i>y</i> | Is the y (row) coordinate of the character's position in the window. | | | | | | | | |
| <i>x</i> | Is the x (column) coordinate of the character's position in the window. | | | | | | | | |
| <i>win</i> | Is a pointer to the window in which the character is to be written. | | | | | | | | |
| DESCRIPTION | <p>The addch() function writes a character to the <code>stdscr</code> window at the current cursor position. The mvaddch() and mvwaddch() functions write the character to the position indicated by the <i>x</i> (column) and <i>y</i> (row) parameters. The mvaddch() function writes the character to the <code>stdscr</code> window, while mvwaddch() writes the character to the window specified by <i>win</i>. The waddch() function is identical to addch(), but writes the character to the window specified by <i>win</i>.</p> <p>These functions advance the cursor after writing the character. Characters that do not fit on the end of the current line are wrapped to the beginning of the next line unless the current line is the last line of the window and scrolling is disabled. In that situation, characters which extend beyond the end of the line are discarded.</p> <p>When <i>ch</i> is a backspace, carriage return, newline, or tab, X/Open Curses moves the cursor appropriately. Each tab character moves the cursor to the next tab stop. By default, tab stops occur every eight columns. When <i>ch</i> is a control character other than backspace, carriage return, newline, or tab, it is written using $\wedge x$ notation, where <i>x</i> is a printable character. When X/Open Curses writes <i>ch</i> to the last character position on a line, it automatically generates a newline. When <i>ch</i> is written to the last character position of a</p> | | | | | | | | |

scrolling region and **scrollok()** is enabled, X/Open Curses scrolls the scrolling region up one line (see **clearok(3XC)**).

RETURN VALUES

On success, these functions return **OK** . Otherwise, they return **ERR** .

ERRORS

None.

SEE ALSO

attroff(3XC) , **bkgdset(3XC)** , **doupdate(3XC)** , **inch(3XC)** ,
insch(3XC) , **nl(3XC)** , **printw(3XC)** , **scrollok(3XC)** , **sclr(3XC)** ,
terminfo(4)

| | | | | | | | | | | | |
|--------------------|--|--------------|--|----------|---|----------|--|----------|---|------------|---|
| NAME | addchstr, addchnstr, mvaddchstr, mvaddchnstr, mvwaddchnstr, mvwaddchstr, waddchstr, waddchnstr – copy a character string (with renditions) to a window | | | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int addchstr(const chtype * chstr); int addchnstr(const chtype * chstr, int n); int mvaddchnstr(int y, int x, const chtype * chstr, int n); int mvaddchstr(int y, int x, const chtype * chstr); int mvwaddchnstr(WINDOW * win, int y, int x, const chtype * chstr, int n); int mvwaddchstr(WINDOW * win, int y, int x, const chtype * chstr); int waddchstr(WINDOW * win, const chtype * chstr); int waddchnstr(WINDOW * win, const chtype * chstr, int n);</pre> | | | | | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="padding-right: 20px;"><i>chstr</i></td> <td>Is a pointer to the <code>chtype</code> string to be copied to the window.</td> </tr> <tr> <td><i>n</i></td> <td>Is the maximum number of characters to be copied from <i>chstr</i>. If <i>n</i> is less than 0, the entire string is written or as much of it as fits on the line.</td> </tr> <tr> <td><i>y</i></td> <td>Is the <i>y</i> (row) coordinate of the starting position of <i>chstr</i> in the window.</td> </tr> <tr> <td><i>x</i></td> <td>Is the <i>x</i> (column) coordinate of the starting position of <i>chstr</i> in the window.</td> </tr> <tr> <td><i>win</i></td> <td>Is a pointer to the window to which the string is to be copied.</td> </tr> </table> | <i>chstr</i> | Is a pointer to the <code>chtype</code> string to be copied to the window. | <i>n</i> | Is the maximum number of characters to be copied from <i>chstr</i> . If <i>n</i> is less than 0, the entire string is written or as much of it as fits on the line. | <i>y</i> | Is the <i>y</i> (row) coordinate of the starting position of <i>chstr</i> in the window. | <i>x</i> | Is the <i>x</i> (column) coordinate of the starting position of <i>chstr</i> in the window. | <i>win</i> | Is a pointer to the window to which the string is to be copied. |
| <i>chstr</i> | Is a pointer to the <code>chtype</code> string to be copied to the window. | | | | | | | | | | |
| <i>n</i> | Is the maximum number of characters to be copied from <i>chstr</i> . If <i>n</i> is less than 0, the entire string is written or as much of it as fits on the line. | | | | | | | | | | |
| <i>y</i> | Is the <i>y</i> (row) coordinate of the starting position of <i>chstr</i> in the window. | | | | | | | | | | |
| <i>x</i> | Is the <i>x</i> (column) coordinate of the starting position of <i>chstr</i> in the window. | | | | | | | | | | |
| <i>win</i> | Is a pointer to the window to which the string is to be copied. | | | | | | | | | | |
| DESCRIPTION | <p>The addchstr() function copies the <code>chtype</code> character string to the <code>stdscr</code> window at the current cursor position. The mvaddchstr() and mvwaddchstr() functions copy the character string to the starting position indicated by the <i>x</i> (column) and <i>y</i> (row) parameters (the former to the <code>stdscr</code> window; the latter to window <i>win</i>). The waddchstr() is identical to addchstr(), but writes to the window specified by <i>win</i>.</p> <p>The addchnstr(), waddchnstr(), mvaddchnstr(), and mvwaddchnstr() functions write <i>n</i> characters to the window, or as many as will fit on the line. If <i>n</i> is less than 0, the entire string is written, or as much of it as fits on the line.</p> | | | | | | | | | | |

The former two functions place the string at the current cursor position; the latter two commands use the position specified by the *x* and *y* parameters.

These functions differ from the `addstr(3XC)` set of functions in two important respects. First, these functions do *not* advance the cursor after writing the string to the window. Second, the current window rendition is not combined with the character; only the attributes that are already part of the `chtype` character are used.

RETURN VALUES

On success, these functions return `OK`. Otherwise, they return `ERR`.

ERRORS

None.

SEE ALSO

`addch(3XC)`, `addnstr(3XC)`, `attroff(3XC)`

| | | | | | | | | | | | |
|--------------------|---|-------------------|---|-----------------|---|-----------------|--|-----------------|---|-------------------|--|
| NAME | addnstr, addstr, mvaddnstr, mvaddstr, mvwaddnstr, mvwaddstr, waddnstr, waddstr – add a multi-byte character string (without rendition) to a window | | | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int addnstr(const char * str, int n); int addstr(const char * str); int mvaddnstr(int y, int x, const char * str, int n); int mvaddstr(int y, int x, const char * str); int mvwaddnstr(WINDOW * win, int y, int x, const char * str, int n); int mvwaddstr(WINDOW * win, int y, int x, const char * str); int waddstr(WINDOW * win, const char * str); int waddnstr(WINDOW * win, const char * str, int n);</pre> | | | | | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="padding-right: 10px;"><i>str</i></td> <td>Is a pointer to the character string that is to be written to the window.</td> </tr> <tr> <td style="padding-right: 10px;"><i>n</i></td> <td>Is the maximum number of characters to be copied from <i>str</i> . If <i>n</i> is less than 0, the entire string is written or as much of it as fits on the line.</td> </tr> <tr> <td style="padding-right: 10px;"><i>y</i></td> <td>Is the <i>y</i> (row) coordinate of the starting position of <i>str</i> in the window.</td> </tr> <tr> <td style="padding-right: 10px;"><i>x</i></td> <td>Is the <i>x</i> (column) coordinate of the starting position of <i>str</i> in the window.</td> </tr> <tr> <td style="padding-right: 10px;"><i>win</i></td> <td>Is a pointer to the window in which the string is to be written.</td> </tr> </table> | <i>str</i> | Is a pointer to the character string that is to be written to the window. | <i>n</i> | Is the maximum number of characters to be copied from <i>str</i> . If <i>n</i> is less than 0, the entire string is written or as much of it as fits on the line. | <i>y</i> | Is the <i>y</i> (row) coordinate of the starting position of <i>str</i> in the window. | <i>x</i> | Is the <i>x</i> (column) coordinate of the starting position of <i>str</i> in the window. | <i>win</i> | Is a pointer to the window in which the string is to be written. |
| <i>str</i> | Is a pointer to the character string that is to be written to the window. | | | | | | | | | | |
| <i>n</i> | Is the maximum number of characters to be copied from <i>str</i> . If <i>n</i> is less than 0, the entire string is written or as much of it as fits on the line. | | | | | | | | | | |
| <i>y</i> | Is the <i>y</i> (row) coordinate of the starting position of <i>str</i> in the window. | | | | | | | | | | |
| <i>x</i> | Is the <i>x</i> (column) coordinate of the starting position of <i>str</i> in the window. | | | | | | | | | | |
| <i>win</i> | Is a pointer to the window in which the string is to be written. | | | | | | | | | | |
| DESCRIPTION | <p>The addstr() function writes a null-terminated string of multi-byte characters to the <code>stdscr</code> window at the current cursor position. The waddstr() function performs an identical action, but writes the character to the window specified by <i>win</i> . The mvaddstr() and mvwaddstr() functions write the string to the position indicated by the <i>x</i> (column) and <i>y</i> (row) parameters (the former to the <code>stdscr</code> window; the latter to window <i>win</i>).</p> <p>The addnstr() , waddnstr() , mvaddnstr() , and mvwaddnstr() functions are similar but write at most <i>n</i> characters to the window. If <i>n</i> is less than 0, the entire string is written.</p> | | | | | | | | | | |

All of these functions advance the cursor after writing the string.

These functions are functionally equivalent to calling the corresponding function from the `addch(3XC)` set of functions once for each character in the string. Refer to the `curses(3XC)` man page for a complete description of special character handling and of the interaction between the window rendition (or background character and rendition) and the character written.

Note that these functions differ from the `addchstr()` set of functions in that the `addchstr(3XC)` functions copy the string as is (without combining each character with the window rendition or the background character and rendition).

RETURN VALUES

On success, these functions return `OK` . Otherwise, they return `ERR` .

ERRORS

None.

SEE ALSO

`addch(3XC)` , `addchstr(3XC)` , `curses(3XC)`

| | | | | | | | | | | | |
|--------------------|---|-------------|--|----------|--|----------|--|----------|---|------------|--|
| NAME | addnwstr, addwstr, mvaddnwstr, mvaddwstr, mvwaddnwstr, mvwaddwstr, waddnwstr, waddwstr – add a wide-character string to a window | | | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int addnwstr(const wchar_t * wstr, int n); int addwstr(const wchar_t * wstr); int mvaddnwstr(int y, int x, const wchar_t * wstr, int n); int mvaddwstr(int y, int x, const wchar_t * wstr); int mvwaddnwstr(WINDOW* win, int y, int x, const wchar_t * wstr, int n); int mvwaddwstr(WINDOW* win, int y, int x, const wchar_t * wstr); int waddnwstr(WINDOW* win, const wchar_t * wstr, int n); int waddwstr(WINDOW* win, const wchar_t * wstr);</pre> | | | | | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="vertical-align: top;">wstr</td> <td>Is a pointer to the wide-character string that is to be written to the window.</td> </tr> <tr> <td style="vertical-align: top;">n</td> <td>Is the maximum number of characters to be copied from <i>wstr</i> . If <i>n</i> is less than 0, the entire string is written or as much of it as fits on the line.</td> </tr> <tr> <td style="vertical-align: top;">y</td> <td>Is the y (row) coordinate of the starting position of <i>wstr</i> in the window.</td> </tr> <tr> <td style="vertical-align: top;">x</td> <td>Is the x (column) coordinate of the starting position of <i>wstr</i> in the window.</td> </tr> <tr> <td style="vertical-align: top;">win</td> <td>Is a pointer to the window in which the string is to be written.</td> </tr> </table> | wstr | Is a pointer to the wide-character string that is to be written to the window. | n | Is the maximum number of characters to be copied from <i>wstr</i> . If <i>n</i> is less than 0, the entire string is written or as much of it as fits on the line. | y | Is the y (row) coordinate of the starting position of <i>wstr</i> in the window. | x | Is the x (column) coordinate of the starting position of <i>wstr</i> in the window. | win | Is a pointer to the window in which the string is to be written. |
| wstr | Is a pointer to the wide-character string that is to be written to the window. | | | | | | | | | | |
| n | Is the maximum number of characters to be copied from <i>wstr</i> . If <i>n</i> is less than 0, the entire string is written or as much of it as fits on the line. | | | | | | | | | | |
| y | Is the y (row) coordinate of the starting position of <i>wstr</i> in the window. | | | | | | | | | | |
| x | Is the x (column) coordinate of the starting position of <i>wstr</i> in the window. | | | | | | | | | | |
| win | Is a pointer to the window in which the string is to be written. | | | | | | | | | | |
| DESCRIPTION | <p>The addwstr() function writes a null-terminated wide-character string to the <code>stdscr</code> window at the current cursor position. The waddwstr() function performs an identical action, but writes the string to the window specified by <i>win</i> . The mvaddwstr() and mvwaddwstr() functions write the string to the position indicated by the <i>x</i> (column) and <i>y</i> (row) parameters (the former to the <code>stdscr</code> window; the latter to window <i>win</i>).</p> <p>The addnwstr() , waddnwstr() , mvaddnwstr() , and mvwaddnwstr() functions write at most <i>n</i> characters to the window. If <i>n</i> is less than 0, the entire string is written. The former two functions place the characters at the</p> | | | | | | | | | | |

current cursor position; the latter two commands use the position specified by the *x* and *y* parameters.

All of these functions advance the cursor after writing the string.

These functions are functionally equivalent to building a `cchar_t` from the `wchar_t` and the window rendition (or background character and rendition) and calling the `wadd_wch(3XC)` function once for each `wchar_t` in the string. Refer to the `curses(3XC)` man page for a complete description of special character handling and of the interaction between the window rendition (or background character and rendition) and the character written.

Note that these functions differ from the `add_wchnstr(3XC)` set of functions in that the latter copy the string as is (without combining each character with the foreground and background attributes of the window).

RETURN VALUES

On success, these functions return `OK` . Otherwise, they return `ERR` .

ERRORS

None.

SEE ALSO

`add_wch(3XC)` , `add_wchnstr(3XC)` , `curses(3XC)`

| NAME | addsev – define additional severities | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <pfmt.h> int addsev(int int_val, const char *string);</pre> | | | | |
| DESCRIPTION | <p>The addsev() function defines additional severities for use in subsequent calls to pfmt(3C) or lfmt(3C). It associates an integer value <i>int_val</i> in the range [5-255] with a character <i>string</i>, overwriting any previous string association between <i>int_val</i> and <i>string</i>.</p> <p>If <i>int_val</i> is OR-ed with the <i>flags</i> argument passed to subsequent calls to pfmt() or lfmt(), <i>string</i> will be used as severity. Passing a null <i>string</i> removes the severity.</p> | | | | |
| RETURN VALUES | Upon successful completion, addsev() returns 0. Otherwise it returns -1. | | | | |
| USAGE | Only the standard severities are automatically displayed for the locale in effect at runtime. An application must provide the means for displaying locale-specific versions of add-on severities. Add-on severities are only effective within the applications defining them. | | | | |
| EXAMPLES | <p>EXAMPLE 1 Example of addsev() function.</p> <p>The following example</p> <pre>#define Panic 5 setlabel("APPL"); setcat("my_appl"); addsev(Panic, gettext(":26", "PANIC")); /* . . . */ lfmt(stderr, MM_SOFT MM_APPL PANIC, ":12:Cannot locate database\n");</pre> <p>will display the message to <i>stderr</i> and forward to the logging service</p> <pre>APPL: PANIC: Cannot locate database</pre> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-safe | | | | |
| SEE ALSO | gettext(3C) , lfmt(3C) , pfmt(3C) , attributes(5) | | | | |

| | | | | | | | | | | | |
|-------------------------|--|----------------------|---|-----------------------|---|-------------------------|---|----------------------|--|-----------------------|---|
| NAME | addseverity – build a list of severity levels for an application for use with <code>fmtmsg</code> | | | | | | | | | | |
| SYNOPSIS | <pre>#include <fmtmsg.h> int addseverity(int severity, const char *string);</pre> | | | | | | | | | | |
| DESCRIPTION | <p>The addseverity() function builds a list of severity levels for an application to be used with the message formatting facility fmtmsg(). The <i>severity</i> argument is an integer value indicating the seriousness of the condition. The <i>string</i> argument is a pointer to a string describing the condition (string is not limited to a specific size).</p> <p>If addseverity() is called with an integer value that has not been previously defined, the function adds that new severity value and print string to the existing set of standard severity levels.</p> <p>If addseverity() is called with an integer value that has been previously defined, the function redefines that value with the new print string. Previously defined severity levels may be removed by supplying the null string. If addseverity() is called with a negative number or an integer value of 0, 1, 2, 3, or 4, the function fails and returns <code>-1</code>. The values 0–4 are reserved for the standard severity levels and cannot be modified. Identifiers for the standard levels of severity are:</p> <table border="0" style="margin-left: 20px;"> <tr> <td style="padding-right: 20px;"><code>MM_HALT</code></td> <td>Indicates that the application has encountered a severe fault and is halting. Produces the print string <code>HALT</code>.</td> </tr> <tr> <td><code>MM_ERROR</code></td> <td>Indicates that the application has detected a fault. Produces the print string <code>ERROR</code>.</td> </tr> <tr> <td><code>MM_WARNING</code></td> <td>Indicates a condition that is out of the ordinary, that might be a problem, and should be watched. Produces the print string <code>WARNING</code>.</td> </tr> <tr> <td><code>MM_INFO</code></td> <td>Provides information about a condition that is not in error. Produces the print string <code>INFO</code>.</td> </tr> <tr> <td><code>MM_NOSEV</code></td> <td>Indicates that no severity level is supplied for the message. Severity levels may also be defined at run time using the <code>SEV_LEVEL</code> environment variable (see fmtmsg(3C)).</td> </tr> </table> | <code>MM_HALT</code> | Indicates that the application has encountered a severe fault and is halting. Produces the print string <code>HALT</code> . | <code>MM_ERROR</code> | Indicates that the application has detected a fault. Produces the print string <code>ERROR</code> . | <code>MM_WARNING</code> | Indicates a condition that is out of the ordinary, that might be a problem, and should be watched. Produces the print string <code>WARNING</code> . | <code>MM_INFO</code> | Provides information about a condition that is not in error. Produces the print string <code>INFO</code> . | <code>MM_NOSEV</code> | Indicates that no severity level is supplied for the message. Severity levels may also be defined at run time using the <code>SEV_LEVEL</code> environment variable (see fmtmsg(3C)). |
| <code>MM_HALT</code> | Indicates that the application has encountered a severe fault and is halting. Produces the print string <code>HALT</code> . | | | | | | | | | | |
| <code>MM_ERROR</code> | Indicates that the application has detected a fault. Produces the print string <code>ERROR</code> . | | | | | | | | | | |
| <code>MM_WARNING</code> | Indicates a condition that is out of the ordinary, that might be a problem, and should be watched. Produces the print string <code>WARNING</code> . | | | | | | | | | | |
| <code>MM_INFO</code> | Provides information about a condition that is not in error. Produces the print string <code>INFO</code> . | | | | | | | | | | |
| <code>MM_NOSEV</code> | Indicates that no severity level is supplied for the message. Severity levels may also be defined at run time using the <code>SEV_LEVEL</code> environment variable (see fmtmsg(3C)). | | | | | | | | | | |
| RETURN VALUES | Upon successful completion, addseverity() returns <code>MM_OK</code> . Otherwise it returns <code>MM_NOTOK</code> . | | | | | | | | | | |
| EXAMPLES | <p>EXAMPLE 1 Example of addseverity() function.</p> <p>When the function call</p> | | | | | | | | | | |

```
addseverity(7,"ALERT")
```

is followed by the call

```
fmtmsg(MM_PRINT, "UX:cat", 7, "invalid syntax", "refer to manual",  
"UX:cat:001")
```

the resulting output is

```
UX:cat: ALERT: invalid syntax  
TO FIX: refer to manual   UX:cat:001
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

fmtmsg(1), **fmtmsg(3C)**, **gettxt(3C)**, **printf(3S)**, **attributes(5)**

| | | | | | | | | | |
|--------------------|--|------------|--|------------|---|----------|--|----------|---|
| NAME | add_wch, mvadd_wch, mvwadd_wch, wadd_wch – add a complex character (with rendition) to a window | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int add_wch(const cchar_t * wch); int wadd_wch(WINDOW * win, const cchar_t * wch); int mvadd_wch(int y, int x, const cchar_t * wch); int mvwadd_wch(WINDOW * win, int y, int x, const cchar_t * wch);</pre> | | | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="vertical-align: top;">wch</td> <td>Is the character/attribute pair (rendition) to be written to the window.</td> </tr> <tr> <td style="vertical-align: top;">win</td> <td>Is a pointer to the window in which the character is to be written.</td> </tr> <tr> <td style="vertical-align: top;">y</td> <td>Is the y (row) coordinate of the character's position in the window.</td> </tr> <tr> <td style="vertical-align: top;">x</td> <td>Is the x (column) coordinate of the character's position in the window.</td> </tr> </table> | wch | Is the character/attribute pair (rendition) to be written to the window. | win | Is a pointer to the window in which the character is to be written. | y | Is the y (row) coordinate of the character's position in the window. | x | Is the x (column) coordinate of the character's position in the window. |
| wch | Is the character/attribute pair (rendition) to be written to the window. | | | | | | | | |
| win | Is a pointer to the window in which the character is to be written. | | | | | | | | |
| y | Is the y (row) coordinate of the character's position in the window. | | | | | | | | |
| x | Is the x (column) coordinate of the character's position in the window. | | | | | | | | |
| DESCRIPTION | <p>The add_wch() function writes a complex character to the <code>stdscr</code> window at the current cursor position. The mvadd_wch() and mvwadd_wch() functions write the character to the position indicated by the <i>x</i> (column) and <i>y</i> (row) parameters. The mvadd_wch() function writes the character to the <code>stdscr</code> window, while mvwadd_wch() writes the character to the window specified by <i>win</i>. The wadd_wch() function is identical to add_wch(), but writes the character to the window specified by <i>win</i>. These functions advance the cursor after writing the character.</p> <p>If <i>wch</i> is a spacing complex character, X/Open Curses replaces any previous character at the specified location with <i>wch</i> (and its rendition). If <i>wch</i> is a non-spacing complex character, X/Open Curses preserves all existing characters at the specified location and adds the non-spacing characters of <i>wch</i> to the spacing complex character. It ignores the rendition associated with <i>wch</i>.</p> <p>Characters that do not fit on the end of the current line are wrapped to the beginning of the next line unless the current line is the last line of the window and scrolling is disabled. In that situation, X/Open Curses discards characters which extend beyond the end of the line.</p> <p>When <i>wch</i> is a backspace, carriage return, newline, or tab, X/Open Curses moves the cursor appropriately as described in the curses(3XC) man page.</p> | | | | | | | | |

Each tab character moves the cursor to the next tab stop. By default, tab stops occur every eight columns. When *wch* is a control character other than a backspace, carriage return, newline, or tab, it is written using `^ x` notation, where *x* is a printable character. When X/Open Curses writes *wch* to the last character position on a line, it automatically generates a newline. When *wch* is written to the last character position of a scrolling region and **scrollok()** is enabled, X/Open Curses scrolls the scrolling region up one line (see **clearok(3XC)**).

RETURN VALUES

On success, these functions return `OK`. Otherwise, they return `ERR`.

ERRORS

None.

SEE ALSO

attr_off(3XC), **bkgrndset(3XC)**, **curses(3XC)**, **doupdate(3XC)**, **in_wch(3XC)**, **ins_wch(3XC)**, **nl(3XC)**, **printw(3XC)**, **scrollok(3XC)**, **scr1(3XC)**, **setscrreg(3XC)**, **terminfo(4)**

| | | | | | | | | | | | |
|--------------------|---|---------------|---|----------|--|----------|---|----------|--|------------|---|
| NAME | add_wchnstr, add_wchstr, mvadd_wchnstr, mvadd_wchstr, mvwadd_wchnstr, mvwadd_wchstr, wadd_wchnstr, wadd_wchstr – copy a string of complex characters (with renditions) to a window | | | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int add_wchnstr(const cchar_t * wchstr, int n); int add_wchstr(const cchar_t * wchstr); int mvadd_wchnstr(int y, int x, const cchar_t * wchstr, int n); int mvadd_wchstr(int y, int x, const cchar_t * wchstr); int mvwadd_wchnstr(WINDOW * win, int y, int x, const cchar_t * wchstr, int n); int mvwaddchstr(WINDOW * win, int y, int x, const cchar_t * wchstr); int wadd_wchstr(WINDOW * win, const cchar_t * wchstr); int wadd_wchnstr(WINDOW * win, const cchar_t * wchstr, int n);</pre> | | | | | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="vertical-align: top;">wchstr</td> <td>Is a pointer to the <code>cchar_t</code> string to be copied to the window.</td> </tr> <tr> <td style="vertical-align: top;">n</td> <td>Is the maximum number of characters to be copied from <code>wchstr</code> . If <code>n</code> is less than 0, the entire string is written or as much of it as fits on the line.</td> </tr> <tr> <td style="vertical-align: top;">y</td> <td>Is the <code>y</code> (row) coordinate of the starting position of <code>wchstr</code> in the window.</td> </tr> <tr> <td style="vertical-align: top;">x</td> <td>Is the <code>x</code> (column) coordinate of the starting position of <code>wchstr</code> in the window.</td> </tr> <tr> <td style="vertical-align: top;">win</td> <td>Is a pointer to the window to which the string is to be copied.</td> </tr> </table> | wchstr | Is a pointer to the <code>cchar_t</code> string to be copied to the window. | n | Is the maximum number of characters to be copied from <code>wchstr</code> . If <code>n</code> is less than 0, the entire string is written or as much of it as fits on the line. | y | Is the <code>y</code> (row) coordinate of the starting position of <code>wchstr</code> in the window. | x | Is the <code>x</code> (column) coordinate of the starting position of <code>wchstr</code> in the window. | win | Is a pointer to the window to which the string is to be copied. |
| wchstr | Is a pointer to the <code>cchar_t</code> string to be copied to the window. | | | | | | | | | | |
| n | Is the maximum number of characters to be copied from <code>wchstr</code> . If <code>n</code> is less than 0, the entire string is written or as much of it as fits on the line. | | | | | | | | | | |
| y | Is the <code>y</code> (row) coordinate of the starting position of <code>wchstr</code> in the window. | | | | | | | | | | |
| x | Is the <code>x</code> (column) coordinate of the starting position of <code>wchstr</code> in the window. | | | | | | | | | | |
| win | Is a pointer to the window to which the string is to be copied. | | | | | | | | | | |
| DESCRIPTION | <p>The <code>add_wchstr()</code> function copies the string of <code>cchar_t</code> characters to the <code>stdscr</code> window at the current cursor position. The <code>mvadd_wchstr()</code> and <code>mvwadd_wchstr()</code> functions copy the string to the starting position indicated by the <code>x</code> (column) and <code>y</code> (row) parameters (the former to the <code>stdscr</code> window; the latter to window <code>win</code>). The <code>wadd_wchstr()</code> is identical to <code>add_wchstr()</code> , but writes to the window specified by <code>win</code> .</p> <p>The <code>add_wchnstr()</code> , <code>wadd_wchnstr()</code> , <code>mvadd_wchnstr()</code> , and <code>mvwadd_wchnstr()</code> functions write <code>n</code> characters to the window, or as many as</p> | | | | | | | | | | |

will fit on the line. If n is less than 0, the entire string is written, or as much of it as fits on the line. The former two functions place the string at the current cursor position; the latter two commands use the position specified by the x and y parameters.

These functions differ from the `addwstr(3XC)` set of functions in two important respects. First, these functions do *not* advance the cursor after writing the string to the window. Second, the current window rendition (that is, the combination of attributes and color pair) is not combined with the character; only those attributes that are already part of the `cchar_t` character are used.

RETURN VALUES

On success, these functions return `OK`. Otherwise, they return `ERR`.

ERRORS

None.

SEE ALSO

`addnwstr(3XC)`, `add_wch(3XC)`, `attr_off(3XC)`

| NAME | aiocancel – cancel an asynchronous operation | | | | |
|----------------------|---|----------------|-----------------|----------|------|
| SYNOPSIS | <pre>cc [flag ...] file ... -laioc [library ...] #include <sys/async.h> int aiocancel(aio_result_t *resultp);</pre> | | | | |
| DESCRIPTION | <p>aiocancel() cancels the asynchronous operation associated with the result buffer pointed to by <i>resultp</i>. It may not be possible to immediately cancel an operation which is in progress and in this case, aiocancel() will not wait to cancel it.</p> <p>Upon successful completion, aiocancel() returns 0 and the requested operation is cancelled. The application will not receive the SIGIO completion signal for an asynchronous operation that is successfully cancelled.</p> | | | | |
| RETURN VALUES | Upon successful completion, aiocancel() returns 0. Upon failure, aiocancel() returns -1 and sets <i>errno</i> to indicate the error. | | | | |
| ERRORS | <p>aiocancel() will fail if any of the following are true:</p> <p>EACCES The parameter <i>resultp</i> does not correspond to any outstanding asynchronous operation, although there is at least one currently outstanding.</p> <p>EFAULT <i>resultp</i> points to an address outside the address space of the requesting process. See NOTES.</p> <p>EINVAL There are not any outstanding requests to cancel.</p> | | | | |
| ATTRIBUTES | See attributes (5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe | | | | |
| SEE ALSO | aioread(3) , aiowait(3) , attributes(5) | | | | |
| NOTES | Passing an illegal address as <i>resultp</i> will result in setting <i>errno</i> to EFAULT <i>only</i> if it is detected by the application process. | | | | |

| | |
|----------------------|---|
| NAME | aio_cancel – cancel asynchronous I/O request |
| SYNOPSIS | <pre>cc [<i>flag...</i>] <i>file...</i> -lrt [<i>library...</i>] #include <aio.h> int aio_cancel(int <i>fd</i>, struct aiocb *<i>aiocbp</i>);</pre> |
| DESCRIPTION | <p>The aio_cancel() function attempts to cancel one or more asynchronous I/O requests currently outstanding against file descriptor <i>fd</i>. The <i>aiocbp</i> argument points to the asynchronous I/O control block for a particular request to be canceled. If <i>aiocbp</i> is NULL, then all outstanding cancelable asynchronous I/O requests against <i>fd</i> are canceled.</p> <p>Normal asynchronous notification occurs for asynchronous I/O operations that are successfully canceled. If there are requests that cannot be canceled, then the normal asynchronous completion process takes place for those requests when they are completed.</p> <p>For requested operations that are successfully canceled, the associated error status is set to ECANCELED and the return status is -1. For requested operations that are not successfully canceled, the <i>aiocbp</i> is not modified by aio_cancel().</p> <p>If <i>aiocbp</i> is not NULL, then if <i>fd</i> does not have the same value as the file descriptor with which the asynchronous operation was initiated, unspecified results occur.</p> |
| RETURN VALUES | <p>The aio_cancel() function returns the value AIO_CANCELED to the calling process if the requested operation(s) were canceled. The value AIO_NOTCANCELED is returned if at least one of the requested operation(s) cannot be canceled because it is in progress. In this case, the state of the other operations, if any, referenced in the call to aio_cancel() is not indicated by the return value of aio_cancel(). The application may determine the state of affairs for these operations by using aio_error(3R). The value AIO_ALLDONE is returned if all of the operations have already completed. Otherwise, the function returns -1 and sets <i>errno</i> to indicate the error.</p> |
| ERRORS | <p>The aio_cancel() function will fail if:</p> <p>EBADF The <i>fd</i> argument is not a valid file descriptor.</p> <p>ENOSYS The aio_cancel() function is not supported.</p> |
| USAGE | <p>The aio_cancel() function has a transitional interface for 64-bit file offsets. See 1f64(5).</p> |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

aio_read(3R), **aio_return(3R)**, **attributes(5)**, **aio(5)**, **lf64(5)**, **signal(5)**

NOTES

Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned `-1` and set `errno` to `ENOSYS`.

| | |
|----------------------|---|
| NAME | aio_error – retrieve errors status for an asynchronous I/O operation |
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <aio.h> int aio_error(const struct aiocb *aiocbp);</pre> |
| DESCRIPTION | The aio_error() function returns the error status associated with the aiocb structure referenced by the aiocbp argument. The error status for an asynchronous I/O operation is the errno value that would be set by the corresponding read(2) , write(2) , or fsync(3C) operation. If the operation has not yet completed, then the error status will be equal to EINPROGRESS . |
| RETURN VALUES | If the asynchronous I/O operation has completed successfully, then 0 is returned. If the asynchronous operation has completed unsuccessfully, then the error status, as described for read(2) , write(2) , and fsync(3C) , is returned. If the asynchronous I/O operation has not yet completed, then EINPROGRESS is returned. |
| ERRORS | <p>The aio_error() function will fail if:</p> <p>ENOSYS The aio_error() function is not supported by the system.</p> <p>The aio_error() function may fail if:</p> <p>EINVAL The aiocbp argument does not refer to an asynchronous operation whose return status has not yet been retrieved.</p> |
| USAGE | The aio_error() function has a transitional interface for 64-bit file offsets. See 1f64(5) . |
| EXAMPLES | <p>EXAMPLE 1 The following is an example of an error handling routine using the aio_error() function.</p> <pre>#include <aio.h> #include <errno.h> #include <signal.h> struct aiocb my_aiocb; struct sigaction my_sigaction; void my_aio_handler(int, siginfo_t *, void *); ... my_sigaction.sa_flags = SA_SIGINFO; my_sigaction.sa_sigaction = my_aio_handler; sigsetempty(&my_sigaction.sa_mask); (void) sigaction(SIGRTMIN, &my_sigaction, NULL); ... my_aiocb.aio_sigevent.sigev_notify = SIGEV_SIGNAL; my_aiocb.aio_sigevent.sigev_signo = SIGRTMIN; my_aiocb.aio_sigevent.sigev_value.sival_ptr = &myaiocb; ... (void) aio_read(&my_aiocb); ... </pre> |

```

void
my_aio_handler(int signo, siginfo_t *siginfo, void *context) {
int my_errno;
struct aiocb *my_aiocbp;

my_aiocbp = siginfo.si_value.sival_ptr;
    if ((my_errno = aio_error(my_aiocb)) != EINPROGRESS) {
        int my_status = aio_return(my_aiocb);
        if (my_status >= 0) { /* start another operation */
            } else { /* handle I/O error */
            }
        }
    }
}

```

ATTRIBUTES

See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------|
| MT-Level | Async-Signal-Safe |

SEE ALSO

[aio_read\(3R\)](#), [aio_write\(3R\)](#), [aio_fsync\(3R\)](#), [lio_listio\(3R\)](#),
[aio_return\(3R\)](#), [aio_cancel\(3R\)](#), [_exit\(2\)](#), [close\(2\)](#), [fork\(2\)](#),
[lseek\(2\)](#), [read\(2\)](#), [write\(2\)](#), [attributes\(5\)](#), [aio\(5\)](#), [lf64\(5\)](#), [signal\(5\)](#)

NOTES

Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned `-1` and set `errno` to `ENOSYS`.

| | |
|--------------------|---|
| NAME | aio_fsync – asynchronous file synchronization |
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <aio.h> int aio_fsync(int op, struct aiocb *aiocb);</pre> |
| DESCRIPTION | <p>The aio_fsync() function asynchronously forces all I/O operations associated with the file indicated by the file descriptor <code>aio_fildes</code> member of the <code>aiocb</code> structure referenced by the <code>aiocbp</code> argument and queued at the time of the call to aio_fsync() to the synchronized I/O completion state. The function call returns when the synchronization request has been initiated or queued to the file or device (even when the data cannot be synchronized immediately).</p> <p>If <code>op</code> is <code>O_DSYNC</code>, all currently queued I/O operations are completed as if by a call to fdatasync(3R); that is, as defined for synchronized I/O data integrity completion. If <code>op</code> is <code>O_SYNC</code>, all currently queued I/O operations are completed as if by a call to fsync(3C); that is, as defined for synchronized I/O file integrity completion. If the aio_fsync() function fails, or if the operation queued by aio_fsync() fails, then, as for fsync(3C) and fdatasync(3R), outstanding I/O operations are not guaranteed to have been completed.</p> <p>If aio_fsync() succeeds, then it is only the I/O that was queued at the time of the call to aio_fsync() that is guaranteed to be forced to the relevant completion state. The completion of subsequent I/O on the file descriptor is not guaranteed to be completed in a synchronized fashion.</p> <p>The <code>aiocbp</code> argument refers to an asynchronous I/O control block. The <code>aiocbp</code> value may be used as an argument to aio_error(3R) and aio_return(3R) in order to determine the error status and return status, respectively, of the asynchronous operation while it is proceeding. When the request is queued, the error status for the operation is <code>EINPROGRESS</code>. When all data has been successfully transferred, the error status will be reset to reflect the success or failure of the operation. If the operation does not complete successfully, the error status for the operation will be set to indicate the error. The <code>aio_sigevent</code> member determines the asynchronous notification to occur when all operations have achieved synchronized I/O completion. All other members of the structure referenced by <code>aiocbp</code> are ignored. If the control block referenced by <code>aiocbp</code> becomes an illegal address prior to asynchronous I/O completion, then the behavior is undefined.</p> <p>If the aio_fsync() function fails or the <code>aiocbp</code> indicates an error condition, data is not guaranteed to have been successfully transferred.</p> <p>If <code>aiocbp</code> is <code>NULL</code>, then no status is returned in <code>aiocbp</code>, and no signal is generated upon completion of the operation.</p> |

RETURN VALUES

The **aio_fsync()** function returns 0 to the calling process if the I/O operation is successfully queued; otherwise, the function returns -1 and sets `errno` to indicate the error.

ERRORS

The **aio_fsync()** function will fail if:

- EAGAIN** The requested asynchronous operation was not queued due to temporary resource limitations.
- EBADF** The `aio_fildes` member of the `aio_cb` structure referenced by the `aio_cbp` argument is not a valid file descriptor open for writing.
- EINVAL** The system does not support synchronized I/O for this file.
- EINVAL** A value of `op` other than `O_DSYNC` or `O_SYNC` was specified.
- ENOSYS** The **aio_fsync()** function is not supported by the system. In the event that any of the queued I/O operations fail, **aio_fsync()** returns the error condition defined for `read(2)` and `write(2)`. The error will be returned in the error status for the asynchronous `fsync(3C)` operation, which can be retrieved using `aio_error(3R)`.

USAGE

The **aio_fsync()** function has a transitional interface for 64-bit file offsets. See `lf64(5)`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`fcntl(2)`, `open(2)`, `read(2)`, `write(2)`, `aio_error(3R)`, `aio_return(3R)`, `fdatasync(3R)`, `fsync(3C)`, `attributes(5)`, `fcntl(5)`, `aio(5)`, `lf64(5)`, `signal(5)`

NOTES

Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned -1 and set `errno` to `ENOSYS`.

| | |
|--------------------|--|
| NAME | aioread, aiowrite – read or write asynchronous I/O operations |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -laio [<i>library</i> ...] #include <sys/types.h> #include <sys/asynch.h> int aioread(int <i>filde</i>s, char * <i>bufp</i>, int <i>bufs</i>, off_t <i>offset</i>, int <i>whence</i>, aio_result_t * <i>resultp</i>); int aiowrite(int <i>filde</i>s, const char * <i>bufp</i>, int <i>bufs</i>, off_t <i>offset</i>, int <i>whence</i>, aio_result_t * <i>resultp</i>); </pre> |
| DESCRIPTION | <p>aioread() initiates one asynchronous read(2) and returns control to the calling program. The read() continues concurrently with other activity of the process. An attempt is made to read <i>bufs</i> bytes of data from the object referenced by the descriptor <i>filde</i>s into the buffer pointed to by <i>bufp</i> .</p> <p>aiowrite() initiates one asynchronous write(2) and returns control to the calling program. The write() continues concurrently with other activity of the process. An attempt is made to write <i>bufs</i> bytes of data from the buffer pointed to by <i>bufp</i> to the object referenced by the descriptor <i>filde</i>s .</p> <p>On objects capable of seeking, the I/O operation starts at the position specified by <i>whence</i> and <i>offset</i> . These parameters have the same meaning as the corresponding parameters to the lseek(2) function. On objects not capable of seeking the I/O operation always start from the current position and the parameters <i>whence</i> and <i>offset</i> are ignored. The seek pointer for objects capable of seeking is not updated by aioread() or aiowrite() . Sequential asynchronous operations on these devices must be managed by the application using the <i>whence</i> and <i>offset</i> parameters.</p> <p>The result of the asynchronous operation is stored in the structure pointed to by <i>resultp</i> :</p> <pre> int aio_return; /* return value of read() or write() */ int aio_errno; /* value of errno for read() or write() */ </pre> |

Upon completion of the operation both *aio_return* and *aio_errno* are set to reflect the result of the operation. `AIO_INPROGRESS` is not a value used by the system so the client may detect a change in state by initializing *aio_return* to this value.

The application supplied buffer *bufp* should not be referenced by the application until after the operation has completed. While the operation is *in progress*, this buffer is in use by the operating system.

Notification of the completion of an asynchronous I/O operation may be obtained synchronously through the `aio_wait(3)` function, or asynchronously by installing a signal handler for the `SIGIO` signal. Asynchronous notification is accomplished by sending the process a `SIGIO` signal. If a signal handler is not installed for the `SIGIO` signal, asynchronous notification is disabled. The delivery of this instance of the `SIGIO` signal is reliable in that a signal delivered while the handler is executing is not lost. If the client ensures that `aio_wait(3)` returns nothing (using a polling timeout) before returning from the signal handler, no asynchronous I/O notifications are lost. The `aio_wait(3)` function is the only way to dequeue an asynchronous notification. Note: `SIGIO` may have several meanings simultaneously: for example, that a descriptor generated `SIGIO` and an asynchronous operation completed. Further, issuing an asynchronous request successfully guarantees that space exists to queue the completion notification.

`close(2)`, `exit(2)` and `execve()` (see `exec(2)`) will block until all pending asynchronous I/O operations can be canceled by the system.

It is an error to use the same result buffer in more than one outstanding request. These structures may only be reused after the system has completed the operation.

RETURN VALUES

Upon successful completion, `aio_read()` and `aio_write()` return 0. Upon failure, `aio_read()` and `aio_write()` return -1 and set `errno` to indicate the error.

ERRORS

`aio_read()` and `aio_write()` will fail if any of the following are true:

| | |
|---------------|--|
| EAGAIN | The number of asynchronous requests that the system can handle at any one time has been exceeded |
| EBADF | <i>files</i> is not a valid file descriptor open for reading. |
| EFAULT | At least one of <i>bufp</i> points to an address outside the address space of the requesting process. See <code>NOTES</code> . |
| EINVAL | The parameter <i>resultp</i> is currently being used by an outstanding asynchronous request. |
| EINVAL | <i>offset</i> is not a valid offset for this file system type. |

ENOMEM Memory resources are unavailable to initiate request.

USAGE The **aioread()** and **aiowrite()** functions have transitional interfaces for 64-bit file offsets. See **lf64(5)** .

ATTRIBUTES See **attributes (5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO **close(2)** , **exec(2)** , **exit(2)** , **llseek(2)** , **lseek(2)** , **open(2)** , **read(2)** , **write(2)** , **aiocancel(3)** , **aiowait(3)** , **sigvec(3B)** , **attributes(5)** , **lf64(5)**

NOTES Passing an illegal address to *bufp* will result in setting **errno** to **EFAULT** *only* if it is detected by the application process.

| | |
|--------------------|---|
| NAME | aio_read – asynchronous read from a file |
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <aio.h> int aio_read(struct aiocb *aiocbp);</pre> |
| DESCRIPTION | <p>The aio_read() function allows the calling process to read <i>aiocbp->aio_nbytes</i> from the file associated with <i>aiocbp->aio_fildes</i> into the buffer pointed to by <i>aiocbp->aio_buf</i>. The function call returns when the read request has been initiated or queued to the file or device (even when the data cannot be delivered immediately). If <code>_POSIX_PRIORITIZED_IO</code> is defined and prioritized I/O is supported for this file, then the asynchronous operation is submitted at a priority equal to the scheduling priority of the process minus <i>aiocbp->aio_reqprio</i>. The <i>aiocbp</i> value may be used as an argument to aio_error(3R) and aio_return(3R) in order to determine the error status and return status, respectively, of the asynchronous operation while it is proceeding. If an error condition is encountered during queuing, the function call returns without having initiated or queued the request. The requested operation takes place at the absolute position in the file as given by <i>aio_offset</i>, as if lseek(2) were called immediately prior to the operation with an <i>offset</i> equal to <i>aio_offset</i> and a <i>whence</i> equal to <code>SEEK_SET</code>. After a successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file is unspecified.</p> <p>The <i>aiocbp->aio_lio_opcode</i> field is ignored by aio_read().</p> <p>The <i>aiocbp</i> argument points to an <code>aiocb</code> structure. If the buffer pointed to by <i>aiocbp->aio_buf</i> or the control block pointed to by <i>aiocbp</i> becomes an illegal address prior to asynchronous I/O completion, then the behavior is undefined.</p> <p>Simultaneous asynchronous operations using the same <i>aiocbp</i> produce undefined results.</p> <p>If <code>_POSIX_SYNCHRONIZED_IO</code> is defined and synchronized I/O is enabled on the file associated with <i>aiocbp->aio_fildes</i>, the behavior of this function is according to the definitions of synchronized I/O data integrity completion and synchronized I/O file integrity completion.</p> <p>For any system action that changes the process memory space while an asynchronous I/O is outstanding to the address range being changed, the result of that action is undefined.</p> <p>For regular files, no data transfer will occur past the offset maximum established in the open file description associated with <i>aiocbp->aio_fildes</i>.</p> |

RETURN VALUES

The **aio_read()** function returns 0 to the calling process if the I/O operation is successfully queued; otherwise, the function returns -1 and sets `errno` to indicate the error.

ERRORS

The **aio_read()** function will fail if:

EAGAIN The requested asynchronous I/O operation was not queued due to system resource limitations.

ENOSYS The **aio_read()** function is not supported by the system. Each of the following conditions may be detected synchronously at the time of the call to **aio_read()**, or asynchronously. If any of the conditions below are detected synchronously, the **aio_read()** function returns -1 and sets `errno` to the corresponding value. If any of the conditions below are detected asynchronously, the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation will be set to the corresponding value.

EBADF The *aio*cbp->aio_fildes argument is not a valid file descriptor open for reading.

EINVAL The file offset value implied by *aio*cbp->aio_offset would be invalid, *aio*cbp->aio_reqprio is not a valid value, or *aio*cbp->aio_nbytes is an invalid value.

In the case that the **aio_read()** successfully queues the I/O operation but the operation is subsequently canceled or encounters an error, the return status of the asynchronous operation is one of the values normally returned by the **read(2)** function call. In addition, the error status of the asynchronous operation will be set to one of the error statuses normally set by the **read()** function call, or one of the following values:

EBADF The *aio*cbp->aio_fildes argument is not a valid file descriptor open for reading.

ECANCELED The requested I/O was canceled before the I/O completed due to an explicit **aio_cancel(3R)** request.

EINVAL The file offset value implied by *aio*cbp->aio_offset would be invalid.

The following condition may be detected synchronously or asynchronously:

E_OVERFLOW The file is a regular file, *aio*cbp->aio_nbytes is greater than 0 and the starting offset in *aio*cbp->aio_offset is before the end-of-file and is at or beyond the offset maximum in the open file description associated with *aio*cbp->aio_fildes.

USAGE

For portability, the application should set *aio*cb->aio_reqprio to 0.

The **aio_read()** function has a transitional interface for 64-bit file offsets. See **lf64(5)**.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

close(2), **exec(2)**, **exit(2)**, **fork(2)**, **lseek(2)**, **read(2)**, **write(2)**, **aio_cancel(3R)**, **aio_return(3R)**, **lio_listio(3R)**, **attributes(5)**, **aio(5)**, **lf64(5)**, **siginfo(5)**, **signal(5)**

NOTES

Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned **-1** and set **errno** to **ENOSYS**.

NAME aio_return - retrieve return status of an asynchronous I/O operation

SYNOPSIS

```
cc [ flag... ] file... -lrt [ library... ]
#include <aio.h>

ssize_t aio_return(struct aiocb *aiocb);
```

DESCRIPTION

The **aio_return()** function returns the return status associated with the `aiocb` structure referenced by the `aiocb` argument. The return status for an asynchronous I/O operation is the value that would be returned by the corresponding `read(2)`, `write(2)`, or `fsync(3C)` function call. If the error status for the operation is equal to `EINPROGRESS`, then the return status for the operation is undefined. The **aio_return()** function may be called exactly once to retrieve the return status of a given asynchronous operation; thereafter, if the same `aiocb` structure is used in a call to **aio_return()** or **aio_error(3R)**, an error may be returned. When the `aiocb` structure referred to by `aiocb` is used to submit another asynchronous operation, then **aio_return()** may be successfully used to retrieve the return status of that operation.

RETURN VALUES

If the asynchronous I/O operation has completed, then the return status, as described for `read(2)`, `write(2)`, and `fsync(3C)`, is returned. If the asynchronous I/O operation has not yet completed, the results of **aio_return()** are undefined.

ERRORS

The **aio_return()** function will fail if:

- EINVAL** The `aiocb` argument does not refer to an asynchronous operation whose return status has not yet been retrieved.
- ENOSYS** The **aio_return()** function is not supported by the system.

USAGE

The **aio_return()** function has a transitional interface for 64-bit file offsets. See `1f64(5)`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------|
| MT-Level | Async-Signal-Safe |

SEE ALSO

`close(2)`, `exec(2)`, `exit(2)`, `fork(2)`, `lseek(2)`, `read(2)`, `write(2)`, `aio_cancel(3R)`, `aio_fsync(3R)`, `aio_read(3R)`, `fsync(3C)`, `lio_listio(3R)`, `attributes(5)`, `aio(5)`, `1f64(5)`, `signal(5)`

NOTES

Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned `-1` and set `errno` to `ENOSYS`.

| | |
|----------------------|---|
| NAME | aio_suspend – wait for asynchronous I/O request |
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <aio.h> int aio_suspend(const struct aiocb * const list[], int nent, const struct timespec *timeout);</pre> |
| DESCRIPTION | <p>The aio_suspend() function suspends the calling thread until at least one of the asynchronous I/O operations referenced by the <i>list</i> argument has completed, until a signal interrupts the function, or, if <i>timeout</i> is not <code>NULL</code>, until the time interval specified by <i>timeout</i> has passed. If any of the <code>aiocb</code> structures in the list correspond to completed asynchronous I/O operations (that is, the error status for the operation is not equal to <code>EINPROGRESS</code>) at the time of the call, the function returns without suspending the calling thread. The <i>list</i> argument is an array of pointers to asynchronous I/O control blocks. The <i>nent</i> argument indicates the number of elements in the array. Each <code>aiocb</code> structure pointed to will have been used in initiating an asynchronous I/O request via aio_read(3R), aio_write(3R), or lio_listio(3R). This array may contain null pointers, which are ignored. If this array contains pointers that refer to <code>aiocb</code> structures that have not been used in submitting asynchronous I/O, the effect is undefined.</p> <p>If the time interval indicated in the <code>timespec</code> structure pointed to by <i>timeout</i> passes before any of the I/O operations referenced by <i>list</i> are completed, then aio_suspend() returns with an error.</p> |
| RETURN VALUES | <p>If aio_suspend() returns after one or more asynchronous I/O operations have completed, it returns 0. Otherwise, it returns -1, and sets <code>errno</code> to indicate the error.</p> <p>The application may determine which asynchronous I/O completed by scanning the associated error and return status using aio_error(3R) and aio_return(3R), respectively.</p> |
| ERRORS | <p>The aio_suspend() function will fail if:</p> <p>EAGAIN No asynchronous I/O indicated in the list referenced by <i>list</i> completed in the time interval indicated by <i>timeout</i>.</p> <p>EINTR A signal interrupted the aio_suspend() function. Note that, since each asynchronous I/O operation may possibly provoke a signal when it completes, this error return may be caused by the completion of one (or more) of the very I/O operations being awaited.</p> <p>ENOSYS The aio_suspend() function is not supported by the system.</p> |

USAGE The **aio_suspend()** function has a transitional interface for 64-bit file offsets. See **lf64(5)**.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------|
| MT-Level | Async-Signal-Safe |

SEE ALSO **aio_fsync(3R)**, **aio_read(3R)**, **aio_return(3R)**, **aio_write(3R)**, **lio_listio(3R)**, **attributes(5)**, **aio(5)**, **lf64(5)**, **signal(5)**

NOTES Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned `-1` and set `errno` to `ENOSYS`.

| NAME | aiowait – wait for completion of asynchronous I/O operation | | | | |
|----------------------|---|----------------|-----------------|----------|------|
| SYNOPSIS | <pre>cc [flag ...] file ... -laio [library ...] #include <sys/asynch.h> #include <sys/time.h></pre> <p><code>aio_result_t *aiowait(const struct timeval *timeout);</code></p> | | | | |
| DESCRIPTION | <p>aiowait() suspends the calling process until one of its outstanding asynchronous I/O operations completes. This provides a synchronous method of notification.</p> <p>If <i>timeout</i> is a non-zero pointer, it specifies a maximum interval to wait for the completion of an asynchronous I/O operation. If <i>timeout</i> is a zero pointer, then aiowait() blocks indefinitely. To effect a poll, the <i>timeout</i> parameter should be non-zero, pointing to a zero-valued <i>timeval</i> structure.</p> <p>The <i>timeval</i> structure is defined in <code><sys/time.h></code> and contains the following members:</p> <pre>long tv_sec; /* seconds */ long tv_usec; /* and microseconds */</pre> | | | | |
| RETURN VALUES | Upon successful completion, aiowait() returns a pointer to the result structure used when the completed asynchronous I/O operation was requested. Upon failure, aiowait() returns <code>-1</code> and sets <code>errno</code> to indicate the error. aiowait() returns <code>0</code> if the time limit expires. | | | | |
| ERRORS | <p>aiowait() will fail if any of the following are true:</p> <p>EFAULT <i>timeout</i> points to an address outside the address space of the requesting process. See NOTES.</p> <p>EINTR aiowait() was interrupted by a signal.</p> <p>EINVAL There are no outstanding asynchronous I/O requests.</p> | | | | |
| ATTRIBUTES | See attributes (5) for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe | | | | |

SEE ALSO | `aiocancel(3)`, `aioread(3)`, `attributes(5)`

NOTES | `aiowait()` is the only way to dequeue an asynchronous notification. It may be used either inside a `SIGIO` signal handler or in the main program. One `SIGIO` signal may represent several queued events.

Passing an illegal address as *timeout* will result in setting `errno` to `EFAULT` *only* if it is detected by the application process.

| | |
|--------------------|--|
| NAME | aio_write – asynchronous write to a file |
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <aio.h> int aio_write(struct aiocb *aiocbp);</pre> |
| DESCRIPTION | <p>The <code>aio_write()</code> function allows the calling process to write <code>aiocbp->aio_nbytes</code> to the file associated with <code>aiocbp->aio_fildes</code> from the buffer pointed to by <code>aiocbp->aio_buf</code>. The function call returns when the write request has been initiated or, at a minimum, queued to the file or device. If <code>_POSIX_PRIORITIZED_IO</code> is defined and prioritized I/O is supported for this file, then the asynchronous operation is submitted at a priority equal to the scheduling priority of the process minus <code>aiocbp->aio_reqprio</code>. The <code>aiocbp</code> may be used as an argument to <code>aio_error(3R)</code> and <code>aio_return(3R)</code> in order to determine the error status and return status, respectively, of the asynchronous operation while it is proceeding.</p> <p>The <code>aiocbp</code> argument points to an <code>aiocb</code> structure. If the buffer pointed to by <code>aiocbp->aio_buf</code> or the control block pointed to by <code>aiocbp</code> becomes an illegal address prior to asynchronous I/O completion, then the behavior is undefined.</p> <p>If <code>O_APPEND</code> is not set for the file descriptor <code>aio_fildes</code>, then the requested operation takes place at the absolute position in the file as given by <code>aio_offset</code>, as if <code>lseek(2)</code> were called immediately prior to the operation with an <code>offset</code> equal to <code>aio_offset</code> and a <code>whence</code> equal to <code>SEEK_SET</code>. If <code>O_APPEND</code> is set for the file descriptor, write operations append to the file in the same order as the calls were made. After a successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file is unspecified.</p> <p>The <code>aiocbp->aio_lio_opcode</code> field is ignored by <code>aio_write()</code>.</p> <p>Simultaneous asynchronous operations using the same <code>aiocbp</code> produce undefined results.</p> <p>If <code>_POSIX_SYNCHRONIZED_IO</code> is defined and synchronized I/O is enabled on the file associated with <code>aiocbp->aio_fildes</code>, the behavior of this function shall be according to the definitions of synchronized I/O data integrity completion and synchronized I/O file integrity completion.</p> <p>For any system action that changes the process memory space while an asynchronous I/O is outstanding to the address range being changed, the result of that action is undefined.</p> <p>For regular files, no data transfer will occur past the offset maximum established in the open file description associated with <code>aiocbp->aio_fildes</code>.</p> |

RETURN VALUES

The **aio_write()** function returns 0 to the calling process if the I/O operation is successfully queued; otherwise, the function returns -1 and sets `errno` to indicate the error.

ERRORS

The **aio_write()** function will fail if:

EAGAIN The requested asynchronous I/O operation was not queued due to system resource limitations.

ENOSYS The **aio_write()** function is not supported by the system. Each of the following conditions may be detected synchronously at the time of the call to **aio_write()**, or asynchronously. If any of the conditions below are detected synchronously, the **aio_write()** function returns -1 and sets `errno` to the corresponding value. If any of the conditions below are detected asynchronously, the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation will be set to the corresponding value.

EBADF The `aiocbp->aio_fildes` argument is not a valid file descriptor open for writing.

EINVAL The file offset value implied by `aiocbp->aio_offset` would be invalid, `aiocbp->aio_reqprio` is not a valid value, or `aiocbp->aio_nbytes` is an invalid value.

In the case that the **aio_write()** successfully queues the I/O operation, the return status of the asynchronous operation will be one of the values normally returned by the `write(2)` function call. If the operation is successfully queued but is subsequently canceled or encounters an error, the error status for the asynchronous operation contains one of the values normally set by the `write()` function call, or one of the following:

EBADF The `aiocbp->aio_fildes` argument is not a valid file descriptor open for writing.

EINVAL The file offset value implied by `aiocbp->aio_offset` would be invalid.

ECANCELED The requested I/O was canceled before the I/O completed due to an explicit `aio_cancel(3R)` request.

The following condition may be detected synchronously or asynchronously:

EFBIG The file is a regular file, `aiocbp->aio_nbytes` is greater than 0 and the starting offset in `aiocbp->aio_offset` is at or beyond the offset maximum in the open file description associated with `aiocbp->aio_fildes`.

USAGE

The **aio_write()** function has a transitional interface for 64-bit file offsets. See `1f64(5)`.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

aio_cancel(3R), **aio_error(3R)**, **aio_read(3R)**, **aio_return(3R)**,
lio_listio(3R), **close(2)**, **_exit(2)**, **fork(2)**, **lseek(2)**, **write(2)**,
attributes(5), **aio(5)**, **lf64(5)**, **signal(5)**

NOTES

Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned `-1` and set `errno` to `ENOSYS`.

| NAME | asin – arc sine function | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | cc [<i>flag ...</i>] <i>file ...</i> -lm [<i>library ...</i>] #include <math.h> double asin (double <i>x</i>); | | | | |
| DESCRIPTION | The asin() function computes the principal value of the arc sine of <i>x</i> . The value of <i>x</i> should be in the range [-1,1]. | | | | |
| RETURN VALUES | Upon successful completion, asin() returns the arc sine of <i>x</i> , in the range [-pi/2,pi/2] radians. If the value of <i>x</i> is not in the range [-1,1] and is not ±Inf or NaN, either 0.0 or NaN is returned and <code>errno</code> is set to EDOM. If <i>x</i> is NaN, NaN is returned. If <i>x</i> is ±Inf, either 0.0 is returned and <code>errno</code> is set to EDOM or NaN is returned and <code>errno</code> may be set to EDOM. For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4. | | | | |
| ERRORS | The asin() function will fail if: EDOM The value <i>x</i> is not ±Inf or NaN and is not in the range [-1,1]. The asin() function may fail if: EDOM The value of <i>x</i> is ±Inf. | | | | |
| USAGE | An application wishing to check for error situations should set <code>errno</code> to 0, then call asin() . If <code>errno</code> is non-zero on return, or the return value is NaN, an error has occurred. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | isnan(3M) , matherr(3M) , sin(3M) , attributes(5) , standards(5) | | | | |

NAME assert – verify program assertion

SYNOPSIS #include <assert.h>

```
void assert(int expression);
```

DESCRIPTION

The **assert()** macro inserts diagnostics into applications. When executed, if *expression* is FALSE (zero), **assert()** prints the error message

```
Assertion failed: expression, file xyz, line nnn
```

on the standard error output and aborts. In the error message, *xyz* is the name of the source file and *nnn* the source line number of the **assert()** statement. These are respectively the values of the preprocessor macros `__FILE__` and `__LINE__`.

Since **assert()** is implemented as a macro, the *expression* may not contain any string literals.

Compiling with the preprocessor option `-DNDEBUG` (see **cc(1B)**), or with the preprocessor control statement `#define NDEBUG` ahead of the `#include <assert.h>` statement, will stop assertions from being compiled into the program.

If the application is linked with `-lintl`, messages printed from this function are in the native language specified by the LC_MESSAGES locale category; see **setlocale(3C)**.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

cc(1B), **abort(3C)**, **gettext(3C)**, **setlocale(3C)**, **attributes(5)**

| | |
|----------------------|--|
| NAME | atan2 – arc tangent function |
| SYNOPSIS | cc [<i>flag ...</i>] <i>file ...</i> -lm [<i>library ...</i>] #include <math.h> double atan2(double y, double x); |
| DESCRIPTION | The atan2() function computes the principal value of the arc tangent of y/x , using the signs of both arguments to determine the quadrant of the return value. |
| RETURN VALUES | Upon successful completion, atan2() returns the arc tangent of y/x in the range $[-\pi, \pi]$ radians. If both arguments are 0.0, 0.0 is returned and <code>errno</code> may be set to EDOM. If x or y is NaN, NaN is returned. In IEEE 754 mode atan2() handles the following exceptional arguments in the spirit of ANSI/IEEE Std 754-1985. <div style="padding-left: 2em;"> <p>atan2($\pm 0, x$) returns ± 0 for $x > 0$ or $x = +0$;</p> <p>atan2($\pm 0, x$) returns $\pm \pi$ for $x < 0$ or $x = -0$;</p> <p>atan2($y, \pm 0$) returns $\pi/2$ for $y > 0$;</p> <p>atan2($y, \pm 0$) returns $-\pi/2$ for $y < 0$;</p> <p>atan2($\pm y, \text{Inf}$) returns ± 0 for finite $y > 0$;</p> <p>atan2($\pm \text{Inf}, x$) returns $\pm \pi/2$ for finite x;</p> <p>atan2($\pm y, -\text{Inf}$) returns $\pm \pi$ for finite $y > 0$;</p> <p>atan2($\pm \text{Inf}, \text{Inf}$) returns $\pm \pi/4$;</p> <p>atan2($\pm \text{Inf}, -\text{Inf}$) returns $\pm 3\pi/4$.</p> </div> <p>For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.</p> |
| ERRORS | The atan2() function may fail if: EDOM Both arguments are 0.0. |

USAGE An application wishing to check for error situations should set `errno` to 0 before calling **atan2()**. If `errno` is non-zero on return, or the return value is NaN, an error has occurred.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **atan(3M)**, **isnan(3M)**, **matherr(3M)**, **tan(3M)**, **attributes(5)**, **standards(5)**

NAME atan – arc tangent function

SYNOPSIS cc [*flag ...*] *file ...* -lm [*library ...*]
#include <math.h>
double **atan**(double *x*);

DESCRIPTION The **atan()** function computes the principal value of the arc tangent of *x*.

RETURN VALUES Upon successful completion, **atan()** returns the arc tangent of *x* in the range $[-\pi/2, \pi/2]$ radians.
If *x* is NaN, NaN is returned.
If *x* is $\pm\text{Inf}$, $\pm\pi/2$ is returned.

ERRORS No errors will occur.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **atan2(3M)**, **isnan(3M)**, **tan(3M)**, **attributes(5)**

NAME atexit – add program termination routine

SYNOPSIS #include <stdlib.h>

```
int atexit(void (*func)(void));
```

DESCRIPTION The **atexit()** function registers the function pointed to by *func* as a list of functions to be called without arguments on normal termination of the program. Normal termination occurs by either a call to the **exit(3C)** function or a return from **main()**. At most 32 functions may be registered by **atexit()**; the functions will be called in the reverse order of their registration.

RETURN VALUES The **atexit()** function returns 0 if the registration succeeds. Otherwise it returns -1.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO **exit(3C)**, **attributes(5)**

| | | | | | | | | | |
|--------------------|--|--------------|--|--------------|---------------------------------------|-------------|-----------------------------|------------|---|
| NAME | attr_get, attr_off, attr_on, attr_set, color_set, wattr_get, wattr_off, wattr_on, wattr_set, wcolor_set – control window attributes | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int attr_get(attr_t * attrs, short * color, void * opts); int attr_off(attr_t attrs, void * opts); int attr_on(attr_t attrs, void * opts); int attr_set(attr_t attrs, short color, void * opts); int color_set(short * color, void * opts); int wattr_get(WINDOW * win, attr_t attrs, short * color, void * opts); int wattr_off(WINDOW * win, attr_t attrs, void * opts); int wattr_on(WINDOW * win, attr_t attrs, void * opts); int wattr_set(WINDOW * win, attr_t attrs, short color, void * opts); int wcolor_set(WINDOW * win, short color, void * opts);</pre> | | | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="padding-right: 20px;">attrs</td> <td>Is a pointer to the foreground window attributes to be set or unset.</td> </tr> <tr> <td>color</td> <td>Is a pointer to a color pair number .</td> </tr> <tr> <td>opts</td> <td>Is reserved for future use.</td> </tr> <tr> <td>win</td> <td>Is a pointer to the window in which attribute changes are to be made.</td> </tr> </table> | attrs | Is a pointer to the foreground window attributes to be set or unset. | color | Is a pointer to a color pair number . | opts | Is reserved for future use. | win | Is a pointer to the window in which attribute changes are to be made. |
| attrs | Is a pointer to the foreground window attributes to be set or unset. | | | | | | | | |
| color | Is a pointer to a color pair number . | | | | | | | | |
| opts | Is reserved for future use. | | | | | | | | |
| win | Is a pointer to the window in which attribute changes are to be made. | | | | | | | | |
| DESCRIPTION | <p>The attr_get() function retrieves the current rendition of <i>stdscr</i> . The wattr_get() function retrieves the current rendition of window <i>win</i> . If <i>attrs</i> or <i>color</i> is a null pointer, no information is retrieved.</p> <p>The attr_off() and attr_on() functions unset and set, respectively, the specified window attributes of <i>stdscr</i> . These functions only affect the attributes specified; attributes that existed before the call are retained.</p> <p>The wattr_off() and wattr_on() functions unset or set the specified attributes for window <i>win</i> .</p> <p>The attr_set() and wattr_set() functions change the rendition of <i>stdscr</i> and <i>win</i> ; the old values are not retained.</p> <p>The color_set() and wcolor_set() functions set the window color of <i>stdscr</i> and <i>win</i> to <i>color</i> .</p> | | | | | | | | |

The attributes and color pairs that can be used are specified in the *Attributes, Color Pairs, and Renditions* section of the `curses(3XC)` man page.

RETURN VALUES

These functions always return `OK` .

ERRORS

None.

SEE ALSO

`add_wch(3XC)` , `addnwstr(3XC)` , `attroff(3XC)` , `bkgrndset(3XC)` , `curses(3XC)` , `init_color(3XC)` , `start_color(3XC)`

| | |
|--------------------|---|
| NAME | attroff, attron, attrset, wattroff, wattron, wattrset – change foreground window attributes |
| SYNOPSIS | <pre>#include <curses.h> int attroff(int attrs); int attron(int attrs); int attrset(int attrs); int wattroff(WINDOW * win, int attrs); int wattron(WINDOW * win, int attrs); int wattrset(WINDOW * win, int attrs);</pre> |
| PARAMETERS | <p>attrs are the foreground window attributes to be set or unset.</p> <p>win Is a pointer to the window in which attribute changes are to be made.</p> |
| DESCRIPTION | <p>The attroff() and attron() functions unset and set, respectively, the specified window attributes of <code>stdscr</code>. These functions only affect the attributes specified; attributes that existed before the call are retained. The wattroff() and wattron() functions unset or set the specified attributes for window <code>win</code>.</p> <p>The attrset() and wattrset() functions change the specified window renditions of <code>stdscr</code> and <code>win</code> to new values; the old values are not retained.</p> <p>The attributes that can be used are specified in the <code>Attributes, Color Pairs, and Renditions</code> section of the curses(3XC) man page.</p> <p>Here is an example that prints some text using the current window rendition, adds underlining, changes the attributes, prints more text, then changes the attributes back.</p> <pre>printw("This word is"); attron(A_UNDERLINE); printw("underlined."); attroff(A_NORMAL); printw("This is back to normal text.\n"); refresh();</pre> |

| | |
|----------------------|---|
| USAGE | All of these functions may be macros. |
| RETURN VALUES | These functions always return OK or 1. |
| ERRORS | None. |
| SEE ALSO | <code>addch(3XC)</code> , <code>addnstr(3XC)</code> , <code>attr_get(3XC)</code> , <code>bkgdset(3XC)</code> , <code>curses(3XC)</code> , <code>init_color(3XC)</code> , <code>start_color(3XC)</code> |

| | |
|----------------------|---|
| NAME | au_open, au_close, au_write – construct and write audit records |
| SYNOPSIS | <pre> cc [flag ...] file ... -lbsm -lsocket -lns1 -lintl [library ...] #include <bsm/libbsm.h> int au_close(int d, int keep, short event); int au_open(void); int au_write(int d, token_t * m); </pre> |
| DESCRIPTION | <p>au_open() returns an audit record descriptor to which audit tokens can be written using au_write(). The audit record descriptor is an integer value that identifies a storage area where audit records are accumulated.</p> <p>au_close() terminates the life of an audit record <i>d</i> of type <i>event</i> started by au_open(). If the <i>keep</i> parameter is zero, the data contained therein is discarded and the memory used is given up by calling free(3C). Otherwise, the additional parameters are used to create a header token. Depending on the audit policy information obtained by auditon(2), additional tokens such as <i>sequence</i> and <i>trailer</i> tokens may be added to the record. au_close() finally writes the record to the audit trail by calling audit(2).</p> <p>au_write() adds the audit token pointed to by <i>m</i> to the audit record identified by the descriptor <i>d</i>. After this call is made the audit token is no longer available to the caller.</p> |
| RETURN VALUES | <p>A successful invocation of au_write() and au_close() will return a 0.</p> <p>A successful invocation of au_open() returns an audit record descriptor. au_open() returns -1 if a descriptor could not be allocated. au_write() returns -1 if <i>d</i> is not a valid descriptor or if audit(2) experienced an error. <i>errno</i> is</p> |

set to indicate the error. **au_write()** will return `-1` if `d` is an invalid descriptor or if `m` is an invalid token.

ATTRIBUTES

See **attributes** (5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

bsmconv(1M), **audit(2)**, **auditon(2)**, **au_preselect(3)**, **au_to(3)**, **free(3C)**, **attributes(5)**

NOTES

The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See **bsmconv(1M)** for more information.

| | | | | | | | | | | | |
|----------------------|---|----------------|--|----------------|--|-------------|--|---------------|--|-----------------|--|
| NAME | au_preselect – preselect an audit event | | | | | | | | | | |
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file...</i> -lbsm -lsocket -lnsl -lintl [<i>library ...</i>] #include <bsm/libbsm.h></pre> <pre>int au_preselect(au_event_t <i>event</i>, au_mask_t *<i>mask_p</i>, int <i>sorf</i>, int <i>flag</i>);</pre> | | | | | | | | | | |
| DESCRIPTION | <p>au_preselect() determines whether or not the audit event <i>event</i> is preselected against the binary preselection mask pointed to by <i>mask_p</i> (usually obtained by a call to getaudit(2)). au_preselect() looks up the classes associated with <i>event</i> in audit_event(4) and compares them with the classes in <i>mask_p</i>. If the classes associated with <i>event</i> match the classes in the specified portions of the binary preselection mask pointed to by <i>mask_p</i>, the event is said to be preselected.</p> <p><i>sorf</i> indicates whether the comparison is made with the success portion, the failure portion or both portions of the mask pointed to by <i>mask_p</i>.</p> <p>The following are the valid values of <i>sorf</i>:</p> <table border="0"> <tr> <td style="padding-right: 20px;">AU_PRS_SUCCESS</td> <td>Compare the event class with the success portion of the preselection mask.</td> </tr> <tr> <td>AU_PRS_FAILURE</td> <td>Compare the event class with the failure portion of the preselection mask.</td> </tr> <tr> <td>AU_PRS_BOTH</td> <td>Compare the event class with both the success and failure portions of the preselection mask.</td> </tr> </table> <p><i>flag</i> tells au_preselect() how to read the audit_event(4) database. Upon initial invocation, au_preselect() reads the audit_event(4) database and allocates space in an internal cache for each entry with malloc(3C). In subsequent invocations, the value of <i>flag</i> determines where au_preselect() obtains audit event information. The following are the valid values of <i>flag</i>:</p> <table border="0"> <tr> <td style="padding-right: 20px;">AU_PRS_REREAD</td> <td>Get audit event information by searching the audit_event(4) database.</td> </tr> <tr> <td>AU_PRS_USECACHE</td> <td>Get audit event information from internal cache created upon the initial invocation. This option is much faster.</td> </tr> </table> | AU_PRS_SUCCESS | Compare the event class with the success portion of the preselection mask. | AU_PRS_FAILURE | Compare the event class with the failure portion of the preselection mask. | AU_PRS_BOTH | Compare the event class with both the success and failure portions of the preselection mask. | AU_PRS_REREAD | Get audit event information by searching the audit_event(4) database. | AU_PRS_USECACHE | Get audit event information from internal cache created upon the initial invocation. This option is much faster. |
| AU_PRS_SUCCESS | Compare the event class with the success portion of the preselection mask. | | | | | | | | | | |
| AU_PRS_FAILURE | Compare the event class with the failure portion of the preselection mask. | | | | | | | | | | |
| AU_PRS_BOTH | Compare the event class with both the success and failure portions of the preselection mask. | | | | | | | | | | |
| AU_PRS_REREAD | Get audit event information by searching the audit_event(4) database. | | | | | | | | | | |
| AU_PRS_USECACHE | Get audit event information from internal cache created upon the initial invocation. This option is much faster. | | | | | | | | | | |
| RETURN VALUES | <p>au_preselect() returns:</p> <table border="0"> <tr> <td style="padding-right: 20px;">0</td> <td><i>event</i> is not preselected.</td> </tr> <tr> <td>1</td> <td><i>event</i> is preselected.</td> </tr> </table> | 0 | <i>event</i> is not preselected. | 1 | <i>event</i> is preselected. | | | | | | |
| 0 | <i>event</i> is not preselected. | | | | | | | | | | |
| 1 | <i>event</i> is preselected. | | | | | | | | | | |

-1 An error occurred. **au_preselect()** couldn't allocate memory or couldn't find *event* in the **audit_event(4)** database.

FILES

`/etc/security/audit_class` maps audit class number to audit class names and descriptions

`/etc/security/audit_event` maps audit even number to audit event names and associates

ATTRIBUTES

See **attributes(5)** for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

bsmconv(1M), **getaudit(2)**, **au_open(3)**, **getauclassent(3)**, **getauevent(3)**, **malloc(3C)**, **audit_class(4)**, **audit_event(4)**, **attributes(5)**

NOTES

au_preselect() is normally called prior to constructing and writing an audit record. If the event is not preselected, the overhead of constructing and writing the record can be saved.

The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See **bsmconv(1M)** for more information.

| | |
|-----------------|---|
| NAME | au_to, au_to_arg, au_to_attr, au_to_data, au_to_groups, au_to_in_addr, au_to_ipc, au_to_ipc_perm, au_to_iport, au_to_me, au_to_opaque, au_to_path, au_to_process, au_to_return, au_to_socket, au_to_text – create audit record tokens |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library</i> ...] #include <sys/types.h> #include <sys/vnode.h> #include <netinet/in.h> #include <bsm/libbsm.h> token_t * au_to_arg(char <i>n</i>, char * <i>text</i>, u_long <i>v</i>); token_t * au_to_attr(struct vattr * <i>attr</i>); token_t * au_to_cmd(u_long <i>argc</i>, char ** <i>argv</i>, char ** <i>envp</i>); token_t * au_to_data(char <i>unit_print</i>, char <i>unit_type</i>, char <i>unit_count</i>, char * <i>p</i>); token_t * au_to_groups(int * <i>groups</i>); token_t * au_to_in_addr(struct inaddr * <i>internet_addr</i>); token_t * au_to_iport(u_short_t <i>iport</i>); token_t * au_to_ipc(int <i>id</i>); token_t * au_to_ipc_perm(struct ipc_perm * <i>perm</i>); token_t * au_to_iport(u_short_t <i>iport</i>); token_t * au_to_me(void); token_t * au_to_newgroups(int <i>n</i>, int * <i>groups</i>);</pre> |

```

token_t *au_to_opaque(char * data, short bytes);

token_t * au_to_path(char * path);

token_t * au_to_process(au_id_t auid, uid_t euid, gid_t egid, uid_t ruid, gid_t rgid,
pid_t pid, au_asid_t sid, au_tid_t * tid);

token_t * au_to_return(char number, uint_t value);

token_t * au_to_socket(struct socket * so);

token_t * au_to_subject(au_id_t auid, uid_t euid, gid_t egid, uid_t ruid, gid_t rgid,
pid_t pid, au_asid_t sid, au_tid_t * tid);

token_t * au_to_text(char * text);

```

DESCRIPTION

au_to_arg() formats the data in *v* into an “argument token.” The *n* argument indicates the argument number. The *text* argument is a null terminated string describing the argument.

au_to_attr() formats the data pointed to by *attr* into a “vnode attribute token.”

au_to_data() formats the data pointed to by *p* into an “arbitrary data token.” The *unit_print* parameter determines the preferred display base of the data and is one of AUP_BINARY , AUP_OCTAL , AUP_DECIMAL , AUP_HEX , or AUP_STRING . The *unit_type* parameter defines the basic unit of data and is one of AUR_BYTE , AUR_CHAR , AUR_SHORT , AUR_INT , or AUR_LONG . The *unit_count* parameter specifies the number of basic data units to be used and must be positive.

au_to_groups() formats the array of 16 integers pointed to by *groups* into a “groups token.”

au_to_in_addr() formats the data pointed to by *internet_addr* into an “internet address token.”

au_to_ipc() formats the data in the *id* parameter into an “interprocess communications id token.”

au_to_ipc_perm() formats the data pointed to by *perm* into an “interprocess communications permission token.”

au_to_iport() formats the data pointed to by *iport* into an “ip port address token.”

au_to_me() collects audit information from the current process and creates a “subject token” by calling **au_to_subject()** .

au_to_newgroups() formats the array of *n* integers pointed to by *groups* into a “newgroups token.”

au_to_subject() formats an *audit user ID* (*audit user ID*), an *effective user ID* (*effective user ID*), an *effective group ID* (*effective group ID*), a *real user ID* (*real user ID*), an *real group ID* (*real group ID*), a *process ID* (*process ID*), an *audit session ID* (*audit session ID*), an *audit terminal ID* (*audit terminal ID*), into a “subject token.”

au_to_opaque() formats the *bytes* bytes pointed to by *data* into an “opaque token.” The value of *size* must be positive.

au_to_path() formats the path name pointed to by *path* into a “path token.”

au_to_process() formats an *audit user ID* (*audit user ID*), an *effective user ID* (*effective user ID*), an *effective group ID* (*effective group ID*), a *real user ID* (*real user ID*), a *real group ID* (*real group ID*), a *process ID* (*process ID*), an *audit session ID* (*audit session ID*), and a *audit terminal ID* (*audit terminal ID*), into a “process token.” A process token should be used when the process is the object of an action (ie. when the process is the receiver of a signal).

au_to_return() formats an error number *number* and a return value *value* into a “return value token.”

au_to_socket() format the data pointed to by *so* into a “socket token.”

au_to_text() formats the NULL terminated string pointed to by *text* into a “text token.”

RETURN VALUES

These functions return NULL if memory cannot be allocated to put the resultant token into, or if an error in the input is detected.

ATTRIBUTES

See **attributes(5)** for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

bsmconv(1M), **au_open(3)**, **attributes(5)**

NOTES

The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See **bsmconv(1M)** for more information.

NAME au_user_mask – get user’s binary preselection mask

SYNOPSIS

```
cc [ flag ... ] file ... -lbsm -lsocket -lnsl -lintl [ library ... ]
#include <bsm/libbsm.h>
```

```
int au_user_mask(char *username, au_mask_t *mask_p);
```

DESCRIPTION

au_user_mask() reads the default, system wide audit classes from **audit_control(4)**, combines them with the per-user audit classes from the **audit_user(4)** database, and updates the binary preselection mask pointed to by *mask_p* with the combined value.

The audit flags in the *flags* field of the **audit_control(4)** database and the *always-audit-flags* and *never-audit-flags* from the **audit_user(4)** database represent binary audit classes. These fields are combined by **au_preselect(3)** as follows:

$$\text{mask} = (\text{flags} + \text{always-audit-flags}) - \text{never-audit-flags}$$

au_user_mask() only fails if both the both the **audit_control(4)** and the **audit_user(4)** database entries could not be retrieved. This allows for flexible configurations.

RETURN VALUES

au_user_mask() returns:

- 0 Success.
- 1 Failure. Both the **audit_control(4)** and the **audit_user(4)** database entries could not be retrieved.

FILES

| | |
|-----------------------------|---|
| /etc/security/audit_control | contains default parameters read by the audit daemon, auditd(1M) |
| /etc/security/audit_user | stores per-user audit event mask |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

login(1), **bsmconv(1M)**, **getaudit(2)**, **setaudit(2)**, **au_preselect(3)**, **getacinfo(3)**, **getausernam(3)**, **audit_control(4)**, **audit_user(4)**, **attributes(5)**

NOTES

au_user_mask() should be called by programs like **login(1)** which set a process's preselection mask with **setaudit(2)**. **getaudit(2)** should be used to obtain audit characteristics for the current process.

The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See **bsmconv(1M)** for more information.

| NAME | basename – return the last element of a path name | | | | | | | | |
|----------------------|---|----------------|-----------------|------------|---------|---------|-------|-----|-----|
| SYNOPSIS | #include <libgen.h> char * basename (char * <i>path</i>); | | | | | | | | |
| DESCRIPTION | The basename() function takes the pathname pointed to by <i>path</i> and returns a pointer to the final component of the pathname, deleting any trailing '/' characters. If the string consists entirely of the '/' character, basename() returns a pointer to the string "/". If <i>path</i> is a null pointer or points to an empty string, basename() returns a pointer to the string ".". | | | | | | | | |
| RETURN VALUES | The basename() function returns a pointer to the final component of <i>path</i> . | | | | | | | | |
| USAGE | The basename() function may modify the string pointed to by <i>path</i> , and may return a pointer to static storage that may then be overwritten by a subsequent call to basename() . When compiling multithreaded applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multithreaded applications. | | | | | | | | |
| EXAMPLES | EXAMPLE 1 Examples for Input String and Output String <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Input String</th> <th style="text-align: center;">Output String</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">"/usr/lib"</td> <td style="text-align: center;">"lib"</td> </tr> <tr> <td style="text-align: center;">"/usr/"</td> <td style="text-align: center;">"usr"</td> </tr> <tr> <td style="text-align: center;">"/"</td> <td style="text-align: center;">"/"</td> </tr> </tbody> </table> | Input String | Output String | "/usr/lib" | "lib" | "/usr/" | "usr" | "/" | "/" |
| Input String | Output String | | | | | | | | |
| "/usr/lib" | "lib" | | | | | | | | |
| "/usr/" | "usr" | | | | | | | | |
| "/" | "/" | | | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe | | | | |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | | | |
| MT-Level | MT-Safe | | | | | | | | |
| SEE ALSO | basename(1) , dirname(3C) , attributes(5) | | | | | | | | |

| | |
|----------------------|--|
| NAME | baudrate – return terminal baud rate |
| SYNOPSIS | <pre>#include <curses.h> int baudrate(void);</pre> |
| DESCRIPTION | The baudrate() function returns the terminal's data communication line and output speed in bits per second (for example, 9600). |
| RETURN VALUES | The baudrate() function returns the output speed of the terminal. |
| ERRORS | None. |

| | |
|----------------------|---|
| NAME | beep, flash – activate audio-visual alarm |
| SYNOPSIS | <pre>#include <curses.h> int beep(void); int flash(void);</pre> |
| DESCRIPTION | The beep() and flash() functions produce an audio and visual alarm on the terminal, respectively. If the terminal has the capability, beep() sounds a bell or beep and flash() flashes the screen. One alarm is substituted for another if the terminal does not support the capability called (see terminfo(4) <code>bel</code> and <code>flash</code> capabilities). For example, a call to beep() for a terminal without that capability results in a flash. |
| RETURN VALUES | These functions always return <code>OK</code> . |
| ERRORS | None. |
| SEE ALSO | terminfo(4) |

NAME ber_decode, ber_alloc_t, ber_free, ber_bvdup, ber_init, ber_flatten, ber_get_next, ber_skiptag, ber_peek_tag, ber_scanf, ber_get_int, ber_get_stringa, ber_get_stringal, ber_get_stringb, ber_get_null, ber_get_boolean, ber_get_bitstring, ber_first_element, ber_next_element, ber_bvfree, ber_bvecfree – Basic Encoding Rules library decoding functions

SYNOPSIS

```
cc[
  flag
  ... ]
file
... -lldap[
  library
  ... ]

#include <lber.h>

BerElement * ber_alloc_t(int options);

struct berval * ber_bvdup(struct berval *bv);

void ber_free(BerElement *ber, int freebuf);

BerElement * ber_init(struct berval *bv);

int ber_flatten(BerElement *ber, struct berval **bvPtr);

ber_get_next(Socketbuf *sb, unsigned long *len, char *bv_val);

ber_skip_tag(BerElement **ber, unsigned long *len);

ber_peek_tag(BerElement **ber, unsigned long *len);

ber_get_int(BerElement **ber, long **num);

ber_get_stringb(BerElement **ber, char **buf, unsigned long *len);

ber_get_stringa(BerElement **ber, char ***buf);

ber_get_stringal(BerElement **ber, struct berval ***bv);

ber_get_null(BerElement **ber);

ber_get_boolean(BerElement **ber, int **bool);

ber_get_bitstringa(BerElement **ber, char ***buf, unsigned long *blen);

ber_first_element(BerElement **ber, unsigned long *len, char **cookie);

ber_next_element(BerElement **ber, unsigned long *len, char **cookie);

ber_scanf(BerElement **ber, char *fmt [, arg.. ]);
```

DESCRIPTION

```
ber_bvfree(struct berval **bv);
```

```
ber_bvecfree(struct berval ***bvec);
```

These functions provide a subfunction interface to a simplified implementation of the Basic Encoding Rules of ASN.1. The version of BER these functions support is the one defined for the LDAP protocol. The encoding rules are the same as BER, except that only definite form lengths are used, and bitstrings and octet strings are always encoded in primitive form. In addition, these lightweight BER functions restrict tags and class to fit in a single octet (this means the actual tag must be less than 31). When a "tag" is specified in the descriptions below, it refers to the tag, class, and primitive or constructed bit in the first octet of the encoding. This man page describes the decoding functions in the lber library. See **ber_encode(3N)** for details on the corresponding encoding functions.

Normally, the only functions that need be called by an application are **ber_get_next()** to get the next BER element and **ber_scanf()** to do the actual decoding. In some cases, **ber_peek_tag()** may also need to be called in normal usage. The other functions are provided for those applications that need more control than **ber_scanf()** provides. In general, these functions return the tag of the element decoded, or `-1` if an error occurred.

The **ber_get_next()** function is used to read the next BER element from the given Sockbuf, *sb*. A Sockbuf consists of the descriptor (usually socket, but a file descriptor works just as well) from which to read, and a BerElement structure used to maintain a buffer. On the first call, the *sb_ber* struct should be zeroed. It strips off and returns the leading tag byte, strips off and returns the length of the entire element in *len*, and sets up *ber* for subsequent calls to **ber_scanf()**, and all to decode the element.

The **ber_scanf()** function is used to decode a BER element in much the same way that **scanf(3S)** works. It reads from *ber*, a pointer to a BerElement such as returned by **ber_get_next()**, interprets the bytes according to the format string *fmt*, and stores the results in its additional arguments. The format string contains conversion specifications which are used to direct the interpretation of the BER element. The format string can contain the following characters.

- a Octet string. A char ** should be supplied. Memory is allocated, filled with the contents of the octet string, null-terminated, and returned in the parameter.
- s Octet string. A char * buffer should be supplied, followed by a pointer to an integer initialized to the size of the buffer. Upon return, the null-terminated octet string is put into the buffer, and the integer is set to the actual size of the octet string.

- O Octet string. A struct `ber_val **` should be supplied, which upon return points to a memory allocated struct `berval` containing the octet string and its length. `ber_bvfree()` can be called to free the allocated memory.
- b Boolean. A pointer to an integer should be supplied.
- i Integer. A pointer to an integer should be supplied.
- B Bitstring. A `char **` should be supplied which will point to the memory allocated bits, followed by an unsigned long *, which will point to the length (in bits) of the bitstring returned.
- n Null. No parameter is required. The element is simply skipped if it is recognized.
- v Sequence of octet strings. A `char ***` should be supplied, which upon return points to a memory allocated null-terminated array of `char *`'s containing the octet strings. `NULL` is returned if the sequence is empty.
- V Sequence of octet strings with lengths. A struct `berval ***` should be supplied, which upon return points to a memory allocated, null-terminated array of struct `berval *`'s containing the octet strings and their lengths. `NULL` is returned if the sequence is empty. `ber_bvecfree()` can be called to free the allocated memory.
- x Skip element. The next element is skipped.
- { Begin sequence. No parameter is required. The initial sequence tag and length are skipped.
- } End sequence. No parameter is required and no action is taken.
-]& Begin set. No parameter is required. The initial set tag and length are skipped.
-] End set. No parameter is required and no action is taken. The `ber_get_int()` function tries to interpret the next element as an integer, returning the result in `num`. The tag of whatever it finds is returned on success, `--1` on failure.

The **ber_get_stringb()** function is used to read an octet string into a preallocated buffer. The *len* parameter should be initialized to the size of the buffer, and will contain the length of the octet string read upon return. The buffer should be big enough to take the octet string value plus a terminating NULL byte.

The **ber_get_stringa()** function is used to allocate memory space into which an octet string is read.

The **ber_get_stringal()** function is used to allocate memory space into which an octet string and its length are read. It takes a struct `berval **`, and returns the result in this parameter.

The **ber_get_null()** function is used to read a NULL element. It returns the tag of the element it skips over.

The **ber_get_boolean()** function is used to read a boolean value. It is called the same way that `ber_get_int()` is called.

The **ber_get_bitstringa()** function is used to read a bitstring value. It takes a `char **` which will hold the allocated memory bits, followed by an unsigned `long *`, which will point to the length (in bits) of the bitstring returned.

The **ber_first_element()** function is used to return the tag and length of the first element in a set or sequence. It also returns in *cookie* a magic cookie parameter that should be passed to subsequent calls to **ber_next_element()**, which returns similar information.

`ber_alloc_t()` constructs and returns `BerElement`. A null pointer is returned on error. The options field contains a bitwise-or of options which are to be used when generating the encoding of this `BerElement`. One option is defined and must always be supplied:

```
#define LBER_USE_DER 0x01
```

When this option is present, lengths will always be encoded in the minimum number of octets. Note that this option does not cause values of sets and sequences to be rearranged in tag and byte order, so these functions are not suitable for generating DER output as defined in X.509 and X.680

The `ber_init` function constructs a `BerElement` and returns a new `BerElement` containing a copy of the data in the `bv` argument. `ber_init` returns the null pointer on error.

`ber_free()` frees a `BerElement` which is returned from the API calls `ber_alloc_t()` or `ber_init()`. Each `BerElement` must be freed by the caller. The second argument *freebuf* should always be set to 1 to ensure that the internal buffer used by the BER functions is freed as well as the `BerElement` container itself.

`ber_bvdup()` returns a copy of a *berval*. The *bv_val* field in the returned *berval* points to a different area of memory as the *bv_val* field in the argument *berval*. The null pointer is returned on error (that is, is out of memory).

The `ber_flatten` routine allocates a struct *berval* whose contents are BER encoding taken from the *ber* argument. The *bvPtr* pointer points to the returned *berval*, which must be freed using `ber_bvfree()`. This routine returns 0 on success and -1 on error.

EXAMPLES

EXAMPLE 1 Assume the variable *ber* contains a lightweight BER encoding of the following ASN.1 object:

```

AlmostASearchRequest := SEQUENCE {
    baseObject      DistinguishedName,
    scope           ENUMERATED {
        baseObject      (0),
        singleLevel    (1),
        wholeSubtree   (2)
    },
    derefAliases    ENUMERATED {
        neverDerefAliases (0),
        derefInSearching (1),
        derefFindingBaseObj (2),
        alwaysDerefAliases (3N)
    },
    sizelimit       INTEGER (0 .. 65535),
    timelimit       INTEGER (0 .. 65535),
    attrsOnly       BOOLEAN,
    attributes      SEQUENCE OF AttributeType
}

```

EXAMPLE 2 The element can be decoded using `ber_scanf()` as follows.

```

int    scope, ali, size, time, attrsonly;
char   *dn, **attrs;
if ( ber_scanf( ber, "{aiiiib{v}}", &dn, &scope, &ali,
              &size, &time, &attrsonly, &attrs ) == --1 )
    /* error */
else
    /* success */

```

ERRORS

If an error occurs during decoding, generally these functions return -1.

NOTES

The return values for all of these functions are declared in the `<lber.h>` header file. Some functions may allocate memory which must be freed by the calling application.

ATTRIBUTES

See `attributes(5)` for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|---|
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO**ber_encode(3N)**

Yeong, W., Howes, T., and Hardcastle-Kille, S., "Lightweight Directory Access Protocol", OSI-DS-26, April 1992.

Information Processing - Open Systems Interconnection - Model and Notation - Service Definition - Specification of Basic Encoding Rules for Abstract Syntax Notation One, International Organization for Standardization, International Standard 8825.

| | |
|--------------------|--|
| NAME | ber_encode, ber_alloc, ber_flush, ber_printf, ber_put_int, ber_put_ostring, ber_put_string, ber_put_null, ber_put_boolean, ber_put_bitstring, ber_start_seq, ber_start_set, ber_put_seq, ber_put_set – simplified Basic Encoding Rules library encoding functions |
| SYNOPSIS | <pre> cc[flag ...] file ... -lldap[library ...] #include <lber.h> BerElement *ber_alloc(); ber_flush(Sockbuf *sb, BerElement *ber, int freit); ber_printf(BerElement *ber, char **fmt [, arg...]); ber_put_int(BerElement *ber, long num, char tag); ber_put_ostring(BerElement *ber, char **str, unsigned long len, char tag); ber_put_string(BerElement *ber, char **str, char tag); ber_put_null(BerElement *ber, char tag); ber_put_boolean(BerElement *ber, int bool, char tag); ber_put_bitstring(BerElement *ber, char *str, int blen, char tag); ber_start_seq(BerElement *ber, char tag); ber_start_set(BerElement *ber, char tag); ber_put_seq(BerElement *ber); ber_put_set(BerElement *ber); </pre> |
| DESCRIPTION | <p>These functions provide a subfunction interface to a simplified implementation of the Basic Encoding Rules of ASN.1. The version of BER these functions support is the one defined for the LDAP protocol. The encoding rules are the same as BER, except that only definite form lengths are used, and bitstrings and octet strings are always encoded in primitive form. In addition, these lightweight BER functions restrict tags and class to fit in a single octet (this means the actual tag must be less than 31). When a "tag" is specified in the descriptions below, it refers to the tag, class, and primitive or constructed bit in the first octet of the encoding. This man page describes the encoding functions</p> |

in the lber library. See **ber_decode(3N)** for details on the corresponding decoding functions.

Normally, the only functions that need be called by an application are **ber_alloc()** to allocate a BER element **ber_printf()** to do the actual encoding, and **ber_flush()** to actually write the element. The other functions are provided for those applications that need more control than **ber_printf()** provides. In general, these functions return the length of the element encoded, or -1 if an error occurred.

The **ber_alloc()** function is used to allocate a new BER element. The **ber_flush()** function is used to actually write the element to a socket (or file) descriptor, once it has been fully encoded (using **ber_printf()** and friends). The *sb* structure contains the descriptor and a *BerElement* used for input buffering. Only the *sb_sd* field is relevant to the **ber_flush()** function.

The **ber_printf()** function is used to encode a BER element in much the same way that **sprintf(3S)** works. One important difference, though, is that some state information is kept with the *ber* parameter so that multiple calls can be made to **ber_printf()** to append things to the end of the BER element.

Ber_printf() writes to *ber*, a pointer to a *BerElement* such as returned by **ber_alloc()**. It interprets and formats its arguments according to the format string *fmt*. The format string can contain the following characters:

- b Boolean. An integer parameter should be supplied. A boolean element is output.
- i Integer. An integer parameter should be supplied. An integer element is output.
- B Bitstring. A char * pointer to the start of the bitstring is supplied, followed by the number of bits in the bitstring. A bitstring element is output.
- n Null. No parameter is required. A null element is output.
- o Octet string. A char * is supplied, followed by the length of the string pointed to. An octet string element is output.
- s Octet string. A null-terminated string is supplied. An octet string element is output, not including the trailing NULL octet.
- t Tag. An int specifying the tag to give the next element is provided. This works across calls.

- v Several octet strings. A null-terminated array of char **s is supplied. Note that a construct like '{v}' is required to get an actual SEQUENCE OF octet strings.
- { Begin sequence. No parameter is required.
- } End sequence. No parameter is required.
-]& Begin set. No parameter is required.
-] End set. No parameter is required.

The **ber_put_int()** function writes the integer element *num* to the BER element *ber*.

The **ber_put_boolean()** function writes the boolean value given by *bool* to the BER element.

The **ber_put_bitstring()** function writes *blen* bits starting at *str* as a bitstring value to the given BER element. Note that *blen* is the length in *bits* of the bitstring.

The **ber_put_ostring()** function writes *len* bytes starting at *str* to the BER element as an octet string.

The **ber_put_string()** function writes the null-terminated string (minus the terminating `'\0'`) to the BER element as an octet string.

The **ber_put_null()** function writes a NULL element to the BER element.

The **ber_start_seq()** function is used to start a sequence in the BER element. The **ber_start_set()** function works similarly. The end of the sequence or set is marked by the nearest matching call to **ber_put_seq()** or **ber_put_set()**, respectively.

The **ber_first_element()** function is used to return the tag and length of the first element in a set or sequence. It also returns in *cookie* a magic cookie parameter that should be passed to subsequent calls to **ber_next_element()**, which returns similar information.

EXAMPLES

EXAMPLE 1 Assuming the following variable declarations, and that the variables have been assigned appropriately, an BER encoding of the following ASN.1 object:

```

AlmostASearchRequest := SEQUENCE {
    baseObject      DistinguishedName,
    scope           ENUMERATED {
        baseObject      (0),
        singleLevel     (1),
        wholeSubtree    (2)
    },
    derefAliases    ENUMERATED {

```

```

        neverDerefAliases (0),
        derefInSearching (1),
        derefFindingBaseObj (2),
        alwaysDerefAliases (3N)
    },
    sizelimit    INTEGER (0 .. 65535),
    timelimit    INTEGER (0 .. 65535),
    attrsOnly    BOOLEAN,
    attributes    SEQUENCE OF AttributeType
}

```

can be achieved like so:

```

int    scope, ali, size, time, attrsonly;
char   *dn, **attrs;

/* ... fill in values ... */
if ( (ber = ber_alloc()) == NULLBER )
/* error */

if ( ber_printf( ber, "{siiiib{v}}", dn, scope, ali,
    size, time, attrsonly, attrs ) == --1 )
/* error */
else
/* success */

```

RETURN VALUES

If an error occurs during encoding, generally these functions return -1 .

ATTRIBUTES

See **attributes(5)** for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|---|
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

lber_decode(3N)

Yeong, W., Howes, T., and Hardcastle-Kille, S., "Lightweight Directory Access Protocol", OSI-DS-26, April 1992.

Information Processing - Open Systems Interconnection - Model and Notation - Service Definition - Specification of Basic Encoding Rules for Abstract Syntax Notation One, International Organization for Standardization, International Standard 8825.

NOTES

The return values for all of these functions are declared in the <lber.h> header file.

| NAME | bgets – read stream up to next delimiter | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lgen [<i>library ...</i>] #include <libgen.h> char *bgets(char *buffer, size_t *count, FILE *stream, const char *breakstring);</pre> | | | | |
| DESCRIPTION | <p>bgets() reads characters from <i>stream</i> into <i>buffer</i> until either <i>count</i> is exhausted or one of the characters in <i>breakstring</i> is encountered in the stream. The read data is terminated with a null byte ('\0') and a pointer to the trailing null is returned. If a <i>breakstring</i> character is encountered, the last non-null is the delimiter character that terminated the scan.</p> <p>Note that, except for the fact that the returned value points to the end of the read string rather than to the beginning, the call</p> <pre>bgets(buffer, sizeof buffer, stream, "\n");</pre> <p>is identical to</p> <pre>fgets (buffer, sizeof buffer, stream);</pre> <p>There is always enough room reserved in the buffer for the trailing null.</p> <p>If <i>breakstring</i> is a null pointer, the value of <i>breakstring</i> from the previous call is used. If <i>breakstring</i> is null at the first call, no characters will be used to delimit the string.</p> | | | | |
| RETURN VALUES | NULL is returned on error or end-of-file. Reporting the condition is delayed to the next call if any characters were read but not yet returned. | | | | |
| EXAMPLES | <p>EXAMPLE 1 Example of bgets() function.</p> <pre>#include <libgen.h> char buffer[8]; /* read in first user name from /etc/passwd */ fp = fopen("/etc/passwd", "r"); bgets(buffer, 8, fp, ":");</pre> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | gets(3S) , attributes(5) | | | | |
| NOTES | When compiling multi-thread applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multi-thread applications. | | | | |

| | |
|----------------------|--|
| NAME | bind – bind a name to a socket |
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lsocket -lnsl [<i>library ...</i>] #include <sys/types.h> #include <sys/socket.h></pre> <pre>int bind(int <i>s</i>, const struct sockaddr *<i>name</i>, socklen_t *<i>namelen</i>);</pre> |
| DESCRIPTION | bind() assigns a name to an unnamed socket. When a socket is created with socket(3N) , it exists in a name space (address family) but has no name assigned. bind() requests that the name pointed to by <i>name</i> be assigned to the socket. |
| RETURN VALUES | If the bind is successful, 0 is returned. A return value of -1 indicates an error, which is further specified in the global <code>errno</code> . |
| ERRORS | <p>The bind() call will fail if:</p> <p>EACCES The requested address is protected and the current user has inadequate permission to access it.</p> <p>EADDRINUSE The specified address is already in use.</p> <p>EADDRNOTAVAIL The specified address is not available on the local machine.</p> <p>EBADF <i>s</i> is not a valid descriptor.</p> <p>EINVAL <i>namelen</i> is not the size of a valid address for the specified address family.</p> <p>EINVAL The socket is already bound to an address.</p> <p>ENOSR There were insufficient STREAMS resources for the operation to complete.</p> <p>ENOTSOCK <i>s</i> is a descriptor for a file, not a socket.</p> <p>The following errors are specific to binding names in the UNIX domain:</p> <p>EACCES Search permission is denied for a component of the path prefix of the pathname in <i>name</i>.</p> <p>EIO An I/O error occurred while making the directory entry or allocating the inode.</p> <p>EISDIR A null pathname was specified.</p> |

| | |
|----------------|---|
| ELOOP | Too many symbolic links were encountered in translating the pathname in <i>name</i> . |
| ENOENT | A component of the path prefix of the pathname in <i>name</i> does not exist. |
| ENOTDIR | A component of the path prefix of the pathname in <i>name</i> is not a directory. |
| EROFS | The inode would reside on a read-only file system. |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

`unlink(2)`, `socket(3N)`, `attributes(5)`, `socket(5)`

NOTES

Binding a name in the UNIX domain creates a socket in the file system that must be deleted by the caller when it is no longer needed (using `unlink(2)`).

The rules used in name binding vary between communication domains.

| | |
|----------------------|--|
| NAME | bind – bind a name to a socket |
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lxnet [<i>library ...</i>] #include <sys/socket.h></pre> <p>int bind(int <i>socket</i>, const struct sockaddr *<i>address</i>, socklen_t <i>address_len</i>);</p> |
| DESCRIPTION | <p>The bind() function assigns an <i>address</i> to an unnamed socket. Sockets created with socket(3XN) function are initially unnamed; they are identified only by their address family.</p> <p>The function takes the following arguments:</p> <p>socket Specifies the file descriptor of the socket to be bound.</p> <p>address Points to a <code>sockaddr</code> structure containing the address to be bound to the socket. The length and format of the address depend on the address family of the socket.</p> <p>address_len Specifies the length of the <code>sockaddr</code> structure pointed to by the <i>address</i> argument.</p> <p>The socket in use may require the process to have appropriate privileges to use the bind() function.</p> |
| USAGE | An application program can retrieve the assigned socket name with the getsockname(3XN) function. |
| RETURN VALUES | Upon successful completion, bind() returns 0. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error. |
| ERRORS | <p>The bind() function will fail if:</p> <p>EADDRINUSE The specified address is already in use.</p> <p>EADDRNOTAVAIL The specified address is not available from the local machine.</p> <p>EAFNOSUPPORT The specified address is not a valid address for the address family of the specified socket.</p> <p>EBADF The <i>socket</i> argument is not a valid file descriptor.</p> <p>EFAULT The <i>address</i> argument can not be accessed.</p> <p>EINVAL The socket is already bound to an address, and the protocol does not support binding to a new address; or the socket has been shut down.</p> <p>ENOTSOCK The <i>socket</i> argument does not refer to a socket.</p> |

| | |
|--|--|
| EOPNOTSUPP | The socket type of the specified socket does not support binding to an address. |
| If the address family of the socket is AF_UNIX, then bind() will fail if: | |
| EACCES | A component of the path prefix denies search permission, or the requested name requires writing in a directory with a mode that denies write permission. |
| EDESTADDRREQ | |
| EISDIR | The <i>address</i> argument is a null pointer. |
| EIO | An I/O error occurred. |
| ELOOP | Too many symbolic links were encountered in translating the pathname in <i>address</i> . |
| ENAMETOOLONG | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX characters. |
| ENOENT | A component of the pathname does not name an existing file or the pathname is an empty string. |
| ENOTDIR | A component of the path prefix of the pathname in <i>address</i> is not a directory. |
| EROFS | The name would reside on a read-only filesystem. |
| The bind() function may fail if: | |
| EACCES | The specified address is protected and the current user does not have permission to bind to it. |
| EINVAL | The <i>address_len</i> argument is not a valid length for the address family. |
| EISCONN | The socket is already connected. |
| ENAMETOOLONG | Pathname resolution of a symbolic link produced an intermediate result whose length exceeds PATH_MAX. |
| ENOBUFS | Insufficient resources were available to complete the call. |
| ENOSR | There were insufficient STREAMS resources for the operation to complete. |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

connect(3XN), **getsockname(3XN)**, **listen(3XN)**, **socket(3XN)**,
attributes(5)

| | |
|--------------------|--|
| NAME | bkgd, bkgdset, wbkgd, wbkgdset – set the background character (and rendition) of window |
| SYNOPSIS | <pre>#include <curses.h> int bkgd(chtype ch); int wbkgd(WINDOW * win, chtype ch); void bkgdset(chtype ch); void wbkgdset(WINDOW * win, chtype ch);</pre> |
| PARAMETERS | <p>ch Is a pointer to the background character to be set.</p> <p>win Is a pointer to the window in which the background character is to be set.</p> |
| DESCRIPTION | <p>All characters except space are part of the foreground. The character and its attributes make up a character/rendition pair defined as a <code>chtype</code>. The character is any single-byte value; the attribute consists of highlighting attributes that affect the appearance of the character on the screen (for example, bold, underline, color).</p> <p>The <code>bkgdset()</code> function sets the current background character and rendition for the <code>stdscr</code> window. <code>wbkgdset()</code> sets the current background character and rendition for window <code>win</code>. You must specify the complete character/rendition pair; for example:</p> <pre>bkgdset(A_BOLD COLOR_PAIR(1) ' ');</pre> <p>sets the background rendition to bold with color and the background character to a space. The default background character/rendition pair is</p> <pre>bkgdset(A_NORMAL COLOR_PAIR(0) ' ');</pre> <p>The current background character and rendition are written to the window by the <code>clear(3XC)</code>, <code>erase(3XC)</code>, <code>cltroeol(3XC)</code>, and <code>cltrobot(3XC)</code> sets of functions as well as any other functions that insert blanks. If a background character is not supplied (that is, only a rendition is given), results are undefined.</p> |

RETURN VALUES

The **bkgd()** and **wbkgd()** functions update the entire window (*stdscr* and *win* , respectively) with the supplied background and perform a **wbkgdset()** .

On success, the **bkgd()** and **wbkgd()** functions return `OK` . Otherwise, they return `ERR` .

The **wbkgdset()** and **wbkgdset()** functions do not return a value.

On success, the **getbkgd()** function returns the background character and rendition for the specified window. Otherwise, it returns `ERR` .

ERRORS

None.

SEE ALSO

`addch(3XC)` , `addchstr(3XC)` , `attroff(3XC)` , `bkgrnd(3XC)` , `clear(3XC)` , `clrtoeol(3XC)` , `clrtobot(3XC)` , `erase(3XC)` , `inch(3XC)` , `mvprintw(3XC)`

| | |
|--------------------|---|
| NAME | bkgnd, bkgndset, getbkgnd, wbkgnd, wbkgndset, wgetbkgnd – set or get the background character (and rendition) of window using a complex character |
| SYNOPSIS | <pre>#include <curses.h> int bkgnd(const cchar_t * wch); void bkgndset(const cchar_t * wch); int getbkgnd(cchar_t * wch); int wbkgnd(WINDOW * win, const cchar_t * wch); void wbkgndset(WINDOW * win, const cchar_t * wch); int wgetbkgnd(WINDOW * win, cchar_t * wch);</pre> |
| PARAMETERS | <p>wch Is a pointer to the complex background character to be set.</p> <p>win Is a pointer to the window in which the complex background character is to be set.</p> |
| DESCRIPTION | <p>All characters except space are part of the foreground. The character and its attributes make up a character/rendition pair defined as a <code>chtype</code>. The character is any single-byte value; the attribute consists of highlighting attributes that affect the appearance of the character on the screen (for example, bold, underline, color).</p> <p>If <i>wch</i> is a multicolumn character that cannot fit on the window line, these functions return <code>ERR</code>.</p> <p>The <code>bkgndset()</code> function sets the current background character and rendition for the <code>stdscr</code> window. <code>wbkgndset()</code> sets the current background character and rendition for window <i>win</i>. You must specify the complete character/rendition pair; for example:</p> <pre>bkgndset(A_BOLD COLOR_PAIR(1) ' ');</pre> <p>sets the background rendition to bold with color and the background character to a space. The default background character/rendition pair is</p> <pre>bkgndset(A_NORMAL COLOR_PAIR(0) ' ');</pre> |

The current background character and rendition are written to the window by the `clear(3XC)`, `erase(3XC)`, `clrtoeol(3XC)`, and `clrtobot(3XC)` sets of functions as well as any other functions that insert blanks. If a background character is not supplied (that is, only a rendition is given), results are undefined.

The `bkgrnd()` and `wbkgrnd()` functions update the entire window (`stdscr` and `win`, respectively) with the supplied background and perform a `wbkgrndset()`.

When calling the `bkgrnd()`, `bkgrndset()`, `wbkgrnd()`, or `wbkgrndset()` function, if `wch` is a complex non-spacing character, it is added to the existing complex background character.

The `getbkgrnd()` and `wgetbkgrnd()` functions retrieve the value of the window's background character (with rendition) and store it in the area pointed to by `wch`.

RETURN VALUES

The `bkgrndset()` and `wbkgrndset()` functions do not return a value.

On success, the other functions return `OK`. Otherwise, they return `ERR`.

ERRORS

None.

SEE ALSO

`add_wch(3XC)`, `add_wchnstr(3XC)`, `addch(3XC)`, `addchstr(3XC)`, `attroff(3XC)`, `bkgd(3XC)`, `clear(3XC)`, `clrtoeol(3XC)`, `clrtobot(3XC)`, `erase(3XC)`, `inch(3XC)`, `mvprintw(3XC)`

| | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------|---|-----------|--|-----------|---|-----------|--|-----------|---|-----------|--|-----------|---|-----------|---|-----------|--|------------|---|--------------|--|--------------|---|
| NAME | border, box, wborder – add a single-byte border to a window | | | | | | | | | | | | | | | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int border(chtype <i>ls</i>, chtype <i>rs</i>, chtype <i>ts</i>, chtype <i>bs</i>, chtype <i>tl</i>, chtype <i>tr</i>, chtype <i>bl</i>, chtype <i>br</i>); int wborder(WINDOW * <i>win</i>, chtype <i>ls</i>, chtype <i>rs</i>, chtype <i>ts</i>, chtype <i>bs</i>, chtype <i>tl</i>, chtype <tr, <i="" chtype="">bl, chtype <i>br</i>); int box(WINDOW * <i>win</i>, chtype <i>verch</i>, chtype <i>horch</i>);</tr,></pre> | | | | | | | | | | | | | | | | | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="vertical-align: top;"><i>ls</i></td> <td>Is the character and rendition used for the left side of the border.</td> </tr> <tr> <td style="vertical-align: top;"><i>rs</i></td> <td>Is the character and rendition used for the right side of the border.</td> </tr> <tr> <td style="vertical-align: top;"><i>ts</i></td> <td>Is the character and rendition used for the top of the border.</td> </tr> <tr> <td style="vertical-align: top;"><i>bs</i></td> <td>Is the character and rendition used for the bottom of the border.</td> </tr> <tr> <td style="vertical-align: top;"><i>tl</i></td> <td>Is the character and rendition used for the top-left corner of the border.</td> </tr> <tr> <td style="vertical-align: top;"><i>tr</i></td> <td>Is the character and rendition used for the top-right corner of the border.</td> </tr> <tr> <td style="vertical-align: top;"><i>bl</i></td> <td>Is the character and rendition used for the bottom-left corner of the border.</td> </tr> <tr> <td style="vertical-align: top;"><i>br</i></td> <td>Is the character and rendition used for the bottom-right corner of the border.</td> </tr> <tr> <td style="vertical-align: top;"><i>win</i></td> <td>Is the pointer to the window in which the border or box is to be drawn.</td> </tr> <tr> <td style="vertical-align: top;"><i>verch</i></td> <td>Is the character and rendition used for the left and right columns of the box.</td> </tr> <tr> <td style="vertical-align: top;"><i>horch</i></td> <td>Is the character and rendition used for the top and bottom rows of the box.</td> </tr> </table> | <i>ls</i> | Is the character and rendition used for the left side of the border. | <i>rs</i> | Is the character and rendition used for the right side of the border. | <i>ts</i> | Is the character and rendition used for the top of the border. | <i>bs</i> | Is the character and rendition used for the bottom of the border. | <i>tl</i> | Is the character and rendition used for the top-left corner of the border. | <i>tr</i> | Is the character and rendition used for the top-right corner of the border. | <i>bl</i> | Is the character and rendition used for the bottom-left corner of the border. | <i>br</i> | Is the character and rendition used for the bottom-right corner of the border. | <i>win</i> | Is the pointer to the window in which the border or box is to be drawn. | <i>verch</i> | Is the character and rendition used for the left and right columns of the box. | <i>horch</i> | Is the character and rendition used for the top and bottom rows of the box. |
| <i>ls</i> | Is the character and rendition used for the left side of the border. | | | | | | | | | | | | | | | | | | | | | | |
| <i>rs</i> | Is the character and rendition used for the right side of the border. | | | | | | | | | | | | | | | | | | | | | | |
| <i>ts</i> | Is the character and rendition used for the top of the border. | | | | | | | | | | | | | | | | | | | | | | |
| <i>bs</i> | Is the character and rendition used for the bottom of the border. | | | | | | | | | | | | | | | | | | | | | | |
| <i>tl</i> | Is the character and rendition used for the top-left corner of the border. | | | | | | | | | | | | | | | | | | | | | | |
| <i>tr</i> | Is the character and rendition used for the top-right corner of the border. | | | | | | | | | | | | | | | | | | | | | | |
| <i>bl</i> | Is the character and rendition used for the bottom-left corner of the border. | | | | | | | | | | | | | | | | | | | | | | |
| <i>br</i> | Is the character and rendition used for the bottom-right corner of the border. | | | | | | | | | | | | | | | | | | | | | | |
| <i>win</i> | Is the pointer to the window in which the border or box is to be drawn. | | | | | | | | | | | | | | | | | | | | | | |
| <i>verch</i> | Is the character and rendition used for the left and right columns of the box. | | | | | | | | | | | | | | | | | | | | | | |
| <i>horch</i> | Is the character and rendition used for the top and bottom rows of the box. | | | | | | | | | | | | | | | | | | | | | | |
| DESCRIPTION | The border() and wborder() functions draw a border around the specified window. All parameters must be single-byte characters whose rendition can be | | | | | | | | | | | | | | | | | | | | | | |

expressed using only constants beginning with `ACS_`. A parameter with the value of 0 is replaced by the default value.

Constant Values for Borders

| Parameter | Default Constant | Default Character |
|--------------|------------------|-------------------|
| <i>verch</i> | ACS_VLINE | |
| <i>horch</i> | ACS_HLINE | - |
| <i>ls</i> | ACS_VLINE | |
| <i>rs</i> | ACS_VLINE | |
| <i>ts</i> | ACS_HLINE | - |
| <i>bs</i> | ACS_HLINE | - |
| <i>bl</i> | ACS_BLCORNER | + |
| <i>br</i> | ACS_BRCORNER | + |
| <i>tl</i> | ACS_ULCORNER | + |
| <i>tr</i> | ACS_URCORNER | + |

The call

```
box(
win
,
verch
,
horch
)
```

is a short form for

```
wborder(
win
,
verch
,
verch
,
horch
,
horch
, 0, 0, 0,
```

0)

When the window is boxed, the bottom and top rows and right and left columns overwrite existing text.

RETURN VALUES

On success, these functions return `OK` . Otherwise, they return `ERR` .

ERRORS

None.

SEE ALSO

`add_wch(3XC)` , `addch(3XC)` , `attr_get(3XC)` , `attroff(3XC)` , `border_set(3XC)`

| | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------|---|-----------|--|-----------|---|-----------|--|-----------|---|-----------|--|-----------|---|-----------|---|-----------|--|------------|---|--------------|--|--------------|---|
| NAME | border_set, box_set, wborder_set – use complex characters (and renditions) to draw borders | | | | | | | | | | | | | | | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int border_set(const cchar_t * ls, const cchar_t * rs, const cchar_t * ts, const cchar_t * bs, const cchar_t * tl, const cchar_t * tr, const cchar_t * bl, const cchar_t * br); int wborder_set(WINDOW * win, const cchar_t * ls, const cchar_t * rs, const cchar_t * ts, const cchar_t * bs, const cchar_t * tl, const cchar_t * tr, const cchar_t * bl, const cchar_t * br); int box_set(WINDOW * win, const cchar_t * verch, const cchar_t * horch);</pre> | | | | | | | | | | | | | | | | | | | | | | |
| PARAMETERS | <table border="0" style="width: 100%;"> <tr> <td style="vertical-align: top; padding-right: 10px;"><i>ls</i></td> <td>Is the character and rendition used for the left side of the border.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 10px;"><i>rs</i></td> <td>Is the character and rendition used for the right side of the border.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 10px;"><i>ts</i></td> <td>Is the character and rendition used for the top of the border.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 10px;"><i>bs</i></td> <td>Is the character and rendition used for the bottom of the border.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 10px;"><i>tl</i></td> <td>Is the character and rendition used for the top-left corner of the border.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 10px;"><i>tr</i></td> <td>Is the character and rendition used for the top-right corner of the border.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 10px;"><i>bl</i></td> <td>Is the character and rendition used for the bottom-left corner of the border.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 10px;"><i>br</i></td> <td>Is the character and rendition used for the bottom-right corner of the border.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 10px;"><i>win</i></td> <td>Is the pointer to the window in which the border or box is to be drawn.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 10px;"><i>verch</i></td> <td>Is the character and rendition used for the left and right columns of the box.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 10px;"><i>horch</i></td> <td>Is the character and rendition used for the top and bottom rows of the box.</td> </tr> </table> | <i>ls</i> | Is the character and rendition used for the left side of the border. | <i>rs</i> | Is the character and rendition used for the right side of the border. | <i>ts</i> | Is the character and rendition used for the top of the border. | <i>bs</i> | Is the character and rendition used for the bottom of the border. | <i>tl</i> | Is the character and rendition used for the top-left corner of the border. | <i>tr</i> | Is the character and rendition used for the top-right corner of the border. | <i>bl</i> | Is the character and rendition used for the bottom-left corner of the border. | <i>br</i> | Is the character and rendition used for the bottom-right corner of the border. | <i>win</i> | Is the pointer to the window in which the border or box is to be drawn. | <i>verch</i> | Is the character and rendition used for the left and right columns of the box. | <i>horch</i> | Is the character and rendition used for the top and bottom rows of the box. |
| <i>ls</i> | Is the character and rendition used for the left side of the border. | | | | | | | | | | | | | | | | | | | | | | |
| <i>rs</i> | Is the character and rendition used for the right side of the border. | | | | | | | | | | | | | | | | | | | | | | |
| <i>ts</i> | Is the character and rendition used for the top of the border. | | | | | | | | | | | | | | | | | | | | | | |
| <i>bs</i> | Is the character and rendition used for the bottom of the border. | | | | | | | | | | | | | | | | | | | | | | |
| <i>tl</i> | Is the character and rendition used for the top-left corner of the border. | | | | | | | | | | | | | | | | | | | | | | |
| <i>tr</i> | Is the character and rendition used for the top-right corner of the border. | | | | | | | | | | | | | | | | | | | | | | |
| <i>bl</i> | Is the character and rendition used for the bottom-left corner of the border. | | | | | | | | | | | | | | | | | | | | | | |
| <i>br</i> | Is the character and rendition used for the bottom-right corner of the border. | | | | | | | | | | | | | | | | | | | | | | |
| <i>win</i> | Is the pointer to the window in which the border or box is to be drawn. | | | | | | | | | | | | | | | | | | | | | | |
| <i>verch</i> | Is the character and rendition used for the left and right columns of the box. | | | | | | | | | | | | | | | | | | | | | | |
| <i>horch</i> | Is the character and rendition used for the top and bottom rows of the box. | | | | | | | | | | | | | | | | | | | | | | |

DESCRIPTION

The **border_set()** and **wborder_set()** functions draw a border around the specified window. All parameters must be spacing complex characters with renditions. A parameter which is a null pointer is replaced by the default character.

Constant Values for Borders

| Parameter | Default Constant | Default Character |
|--------------|------------------|-------------------|
| <i>verch</i> | WACS_VLINE | |
| <i>horch</i> | WACS_HLINE | - |
| <i>ls</i> | WACS_VLINE | |
| <i>rs</i> | WACS_VLINE | |
| <i>ts</i> | WACS_HLINE | - |
| <i>bs</i> | WACS_HLINE | - |
| <i>bl</i> | WACS_BLCORNER | + |
| <i>br</i> | WACS_BRCORNER | + |
| <i>tl</i> | WACS_ULCORNER | + |
| <i>tr</i> | WACS_URCORNER | + |

The call

```

box_set(
win
,
verch
,
horch
)

```

is a short form for

```

wborder(
win
,
verch
,
verch
,
horch

```

```
'  
horch  
, NULL,  
NULL, NULL, NULL)
```

When the window is boxed, the bottom and top rows and right and left columns are unavailable for text.

RETURN VALUES

On success, these functions return `OK` . Otherwise, they return `ERR` .

ERRORS

None.

SEE ALSO

`add_wch(3XC)` , `addch(3XC)` , `attr_get(3XC)` , `attroff(3XC)` ,
`border(3XC)`

| | |
|----------------------|---|
| NAME | bsdmalloc – memory allocator |
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lbsdmalloc [<i>library ...</i>]</pre> <pre>char *malloc(<i>size</i>);</pre> <pre>unsigned <i>size</i>;</pre> <pre>int free(<i>ptr</i>);</pre> <pre>char *ptr;</pre> <pre>char *realloc(<i>ptr</i>, <i>size</i>);</pre> <pre>char *ptr;</pre> <pre>unsigned <i>size</i>;</pre> |
| DESCRIPTION | <p>These routines provide a general-purpose memory allocation package. They maintain a table of free blocks for efficient allocation and coalescing of free storage. When there is no suitable space already free, the allocation routines call <code>sbrk(2)</code> to get more memory from the system. Each of the allocation routines returns a pointer to space suitably aligned for storage of any type of object. Each returns a null pointer if the request cannot be completed (see <code>DIAGNOSTICS</code>).</p> <p>malloc() returns a pointer to a block of at least <code>size</code> bytes, which is appropriately aligned.</p> <p>free() releases a previously allocated block. Its argument is a pointer to a block previously allocated by malloc() or realloc().</p> <p>realloc() changes the size of the block referenced by <code>ptr</code> to <code>size</code> bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes. If unable to honor a reallocation request, realloc() leaves its first argument unaltered. For backwards compatibility, realloc() accepts a pointer to a block freed since the most recent call to malloc() or realloc().</p> |
| RETURN VALUES | malloc() and realloc() return a null pointer if there is not enough available memory. When realloc() returns <code>NULL</code> , the block pointed to by <code>ptr</code> is left intact. |
| ERRORS | <p>If malloc() or realloc() returns unsuccessfully, <code>errno</code> will be set to indicate the following:</p> <p>ENOMEM <code>size</code> bytes of memory exceeds the physical limits of your system, and cannot be allocated.</p> <p>EAGAIN There is not enough memory available at this point in time to allocate <code>size</code> bytes of memory; but the application could try again later.</p> |

SEE ALSO `brk(2)`, `malloc(3C)`, `malloc(3X)`, `mapmalloc(3X)`

WARNINGS Use of `libbsdmalloc` renders an application non-SCD compliant.

`libbsdmalloc` routines are incompatible with the memory allocation routines in the standard C-library (`libc`): `malloc(3C)`, `alloca(3C)`, `calloc(3C)`, `free(3C)`, `memalign(3C)`, `realloc(3C)`, and `valloc(3C)`.

NOTES Using `realloc()` with a block freed before the most recent call to `malloc()` or `realloc()` will result in an error.

`malloc()` and `realloc()` return a non-NULL pointer if `size` is 0. These pointers should not be dereferenced.

Always cast the value returned by `malloc()` and `realloc()`.

Comparative features of `bsdmalloc()`, `malloc(3X)`, and `malloc(3C)`

- The `bsdmalloc()` routines afford better performance, but are space-inefficient.
- The `malloc(3X)` routines are space-efficient, but have slower performance.
- The standard, fully SCD-compliant `malloc(3C)` routines are a trade-off between performance and space-efficiency.

`free()` does not set `errno`.

| | |
|----------------------|--|
| NAME | bsd_signal – simplified signal facilities |
| SYNOPSIS | <pre>#include <signal.h> void (*bsd_signal(int sig, void (*func)(int)))(int);</pre> |
| DESCRIPTION | <p>The bsd_signal() function provides a partially compatible interface for programs written to historical system interfaces (see USAGE below).</p> <p>The function call bsd_signal(sig, func) has an effect as if implemented as:</p> <pre>void (*bsd_signal(int sig, void (*func)(int)))(int) { struct sigaction act, oact; act.sa_handler = func; act.sa_flags = SA_RESTART; sigemptyset(&act.sa_mask); sigaddset(&act.sa_mask, sig); if (sigaction(sig, &act, &oact) == -1) return(SIG_ERR); return(oact.sa_handler); }</pre> <p>The handler function should be declared:</p> <pre>void handler(int sig);</pre> <p>where <i>sig</i> is the signal number. The behavior is undefined if <i>func</i> is a function that takes more than one argument, or an argument of a different type.</p> |
| RETURN VALUES | Upon successful completion, bsd_signal() returns the previous action for <i>sig</i> . Otherwise, SIG_ERR is returned and errno is set to indicate the error. |
| ERRORS | Refer to sigaction(2) . |
| USAGE | This function is a direct replacement for the BSD signal(3B) function for simple applications that are installing a single-argument signal handler function. If a BSD signal handler function is being installed that expects more than one argument, the application has to be modified to use sigaction(2) . The bsd_signal() function differs from signal(3B) in that the SA_RESTART flag is set and the SA_RESETHAND will be clear when bsd_signal() is used. The state of these flags is not specified for signal(3B) . |
| SEE ALSO | sigaction(2) , sigaddset(3C) , sigemptyset(3C) , signal(3B) |

| | |
|----------------------|--|
| NAME | bsearch – binary search a sorted table |
| SYNOPSIS | <pre>#include <stdlib.h> void *bsearch(const void *key, const void *base, size_t nel, size_t size, int (*compar)(const void *,const void *));</pre> |
| DESCRIPTION | <p>The bsearch() function is a binary search routine generalized from Knuth (6.2.1) Algorithm B. It returns a pointer into a table (an array) indicating where a datum may be found or a null pointer if the datum cannot be found. The table must be previously sorted in increasing order according to a comparison function pointed to by <i>compar</i>.</p> <p>The <i>key</i> argument points to a datum instance to be sought in the table. The <i>base</i> argument points to the element at the base of the table. The <i>nel</i> argument is the number of elements in the table. The <i>size</i> argument is the number of bytes in each element.</p> <p>The comparison function pointed to by <i>compar</i> is called with two arguments that point to the <i>key</i> object and to an array element, in that order. The function must return an integer less than, equal to, or greater than 0 if the <i>key</i> object is considered, respectively, to be less than, equal to, or greater than the array element.</p> |
| RETURN VALUES | The bsearch() function returns a pointer to a matching member of the array, or a null pointer if no match is found. If two or more members compare equal, which member is returned is unspecified. |
| USAGE | <p>The pointers to the key and the element at the base of the table should be of type pointer-to-element.</p> <p>The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.</p> <p>If the number of elements in the table is less than the size reserved for the table, <i>nel</i> should be the lower number.</p> |
| EXAMPLES | <p>EXAMPLE 1 Examples for searching a table containing pointers to nodes.</p> <p>The example below searches a table containing pointers to nodes consisting of a string and its length. The table is ordered alphabetically on the string in the node pointed to by each entry.</p> <p>This program reads in strings and either finds the corresponding node and prints out the string and its length, or prints an error message.</p> <pre>#include <stdio.h> #include <stdlib.h> #include <string.h></pre> |

```

struct node { /* these are stored in the table */
    char *string;
    int length;
};
static struct node table[] = { /* table to be searched */
    { "asparagus", 10 },
    { "beans", 6 },
    { "tomato", 7 },
    { "watermelon", 11 },
};

main()
{
    struct node *node_ptr, node;
    /* routine to compare 2 nodes */
    static int node_compare(const void *, const void *);
    char str_space[20]; /* space to read string into */

    node.string = str_space;
    while (scanf("%20s", node.string) != EOF) {
        node_ptr = bsearch( &node,
            table, sizeof(table)/sizeof(struct node),
            sizeof(struct node), node_compare);
        if (node_ptr != NULL) {
            (void) printf("string = %20s, length = %d\n",
                node_ptr->string, node_ptr->length);
        } else {
            (void) printf("not found: %20s\n", node.string);
        }
    }
    return(0);
}

/* routine to compare two nodes based on an */
/* alphabetical ordering of the string field */
static int
node_compare(const void *node1, const void *node2) {
    return (strcmp(
        ((const struct node *)node1)->string,
        ((const struct node *)node2)->string));
}

```

ATTRIBUTES

See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

[hsearch\(3C\)](#), [lsearch\(3C\)](#), [qsort\(3C\)](#), [tsearch\(3C\)](#), [attributes\(5\)](#)

| | |
|--------------------|---|
| NAME | bstring, bcopy, bcmp, bzero – bit and byte string operations |
| SYNOPSIS | <pre>#include <strings.h> void bcopy(const void * <i>s1</i>, void * <i>s2</i>, size_t <i>n</i>); int bcmp(const void * <i>s1</i>, const void * <i>s2</i>, size_t <i>n</i>); void bzero(void * <i>s</i>, size_t <i>n</i>);</pre> |
| DESCRIPTION | <p>The bcopy() , bcmp() , and bzero() functions operate on variable length strings of bytes. They do not check for null bytes as do the functions described on the string(3C) manual page.</p> <p>The bcopy() function copies <i>n</i> bytes from string <i>s1</i> to the string <i>s2</i> . Overlapping strings are handled correctly.</p> <p>The bcmp() function compares byte string <i>s1</i> against byte string <i>s2</i> , returning 0 if they are identical, 1 otherwise. Both strings are assumed to be <i>n</i> bytes long. The bcmp() function using <i>n</i> zero bytes always returns 0 .</p> <p>The bzero() function places <i>n</i> 0 bytes in the string <i>s</i> .</p> |
| WARNINGS | The bcmp() and bcopy() routines take parameters backwards from strcmp() and strcpy() , respectively. See string(3C) . |
| SEE ALSO | memory(3C) , string(3C) |

| NAME | btowc – single-byte to wide-character conversion | | | | |
|----------------------|---|----------------|-----------------|----------|-------------------------|
| SYNOPSIS | <pre>#include <stdio.h> #include <wchar.h> wint_t btowc(int c);</pre> | | | | |
| DESCRIPTION | <p>The btowc() function determines whether <i>c</i> constitutes a valid (one-byte) character in the initial shift state.</p> <p>The behavior of this function is affected by the LC_CTYPE category of the current locale. See environ(5).</p> | | | | |
| RETURN VALUES | <p>The btowc() function returns WEOF if <i>c</i> has the value EOF or if (unsigned char)<i>c</i> does not constitute a valid (one-byte) character in the initial shift state. Otherwise, it returns the wide-character representation of that character.</p> | | | | |
| ERRORS | No errors are defined. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe with exceptions</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe with exceptions |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe with exceptions | | | | |
| SEE ALSO | setlocale(3C) , wctob(3C) , attributes(5) , environ(5) | | | | |
| NOTES | The btowc() function can be used safely in multithreaded applications, as long as setlocale(3C) is not being called to change the locale. | | | | |

| NAME | bufsplit – split buffer into fields | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lgen [<i>library ...</i>] #include <libgen.h> size_t bufsplit(char *<i>buf</i>, size_t <i>n</i>, char **<i>a</i>);</pre> | | | | |
| DESCRIPTION | <p>bufsplit() examines the buffer, <i>buf</i>, and assigns values to the pointer array, <i>a</i>, so that the pointers point to the first <i>n</i> fields in <i>buf</i> that are delimited by TABs or NEWLINES.</p> <p>To change the characters used to separate fields, call bufsplit() with <i>buf</i> pointing to the string of characters, and <i>n</i> and <i>a</i> set to zero. For example, to use colon (:), period (.), and comma (,), as separators along with TAB and NEWLINE:</p> <pre>bufsplit (":.,\t\n", 0, (char**)0);</pre> | | | | |
| RETURN VALUES | The number of fields assigned in the array <i>a</i> . If <i>buf</i> is zero, the return value is zero and the array is unchanged. Otherwise the value is at least one. The remainder of the elements in the array are assigned the address of the null byte at the end of the buffer. | | | | |
| EXAMPLES | <p>EXAMPLE 1 Example of bufsplit() function.</p> <pre>/* * set a[0] = "This", a[1] = "is", a[2] = "a", * a[3] = "test" */ bufsplit("This\tis\ta\ttest\n", 4, a);</pre> | | | | |
| NOTES | <p>bufsplit() changes the delimiters to null bytes in <i>buf</i>.</p> <p>When compiling multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the compile line. This flag should only be used in multithreaded applications.</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | attributes(5) | | | | |

NAME byteorder, htonl, htons, ntohl, ntohs – convert values between host and network byte order

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
#include <inttypes.h>
```

```
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

DESCRIPTION

These routines convert 16 and 32 bit quantities between network byte order and host byte order. On some architectures these routines are defined as NULL macros in the include file `<netinet/in.h>`. On other architectures, if their host byte order is different from network byte order, these routines are functional.

These routines are most often used in conjunction with Internet addresses and ports as returned by `gethostent()` and `getservent()`. See `gethostbyname(3N)` and `getservbyname(3N)`.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO `gethostbyname(3N)`, `getservbyname(3N)`, `attributes(5)`, `inet(5)`

NAME cancellation – overview of concepts related to POSIX thread cancellation

DESCRIPTION

| FUNCTION | ACTION |
|-------------------------------------|---|
| <code>pthread_cancel</code> | Cancels thread execution. |
| <code>pthread_setcancelstate</code> | Sets the cancellation <i>state</i> of a thread. |
| <code>pthread_setcanceltype</code> | Sets the cancellation <i>type</i> of a thread. |
| <code>pthread_testcancel</code> | Creates a cancellation point in the calling thread. |
| <code>pthread_cleanup_push</code> | Pushes a cleanup handler routine. |
| <code>pthread_cleanup_pop</code> | Pops a cleanup handler routine. |

Cancellation

Thread cancellation allows a thread to terminate the execution of any application thread in the process. Cancellation is useful when further operations of one or more threads are undesirable or unnecessary.

An example of a situation that could benefit from using cancellation is an asynchronously-generated cancel condition such as a user requesting to close or exit some running operation. Another example is the completion of a task undertaken by a number of threads, such as solving a maze. While many threads search for the solution, one of the threads might solve the puzzle while the others continue to operate. Since they are serving no purpose at that point, they should all be canceled.

Planning Steps

Planning and programming for most cancellations follow this pattern:

1. Identify which threads you want to cancel, and insert `pthread_cancel(3T)` statements.
2. Identify system-defined cancellation points where a thread that might be canceled could have changed system or program state that should be restored. See the Cancellation Points for a list.
3. When a thread changes the system or program state just before a cancellation point, and should restore that state before the thread is canceled, place a cleanup handler before the cancellation point with `pthread_cleanup_push(3T)`. Wherever a thread restores the changed state, pop the cleanup handler from the cleanup stack with `pthread_cleanup_pop(3T)`.
4. Know whether the threads you are canceling call into cancel-unsafe libraries, and disable cancellation with `pthread_setcancelstate(3T)` before the call into the library. See Cancellation State and Cancel-Safe.

5. To cancel a thread in a procedure that contains no cancellation points, insert your own cancellation points with `pthread_testcancel(3T)`. `pthread_testcancel(3T)` creates cancellation points by testing for pending cancellations and performing those cancellations if they are found. Push and pop cleanup handlers around the cancellation point, if necessary (see Step 3, above).

Cancellation Points

The system defines certain points at which cancellation can occur (cancellation points), and you can create additional cancellation points in your application with `pthread_testcancel(3T)`.

The following cancellation points are defined by the system (system-defined cancellation points): `aio_suspend(3R)`, `close(2)`, `creat(2)`, `getmsg(2)`, `getpmsg(2)`, `lockf(3C)`, `mq_receive(3R)`, `mq_send(3R)`, `msgrcv(2)`, `msgsnd(2)`, `msync(3C)`, `nanosleep(3R)`, `open(2)`, `pause(2)`, `poll(2)`, `pread(2)`, `pthread_cond_timedwait(3T)`, `pthread_cond_wait(3T)`, `pthread_join(3T)`, `pthread_testcancel(3T)`, `putmsg(2)`, `putpmsg(2)`, `pwrite(2)`, `read(2)`, `readv(2)`, `select(3C)`, `sem_wait(3R)`, `sigpause(3C)`, `sigwaitinfo(3R)`, `sigsuspend(2)`, `sigtimedwait(3R)`, `sigwait(2)`, `sleep(3C)`, `sync(3C)`, `system(3S)`, `tcdrain(3)`, `usleep(3C)`, `wait(2)`, `waitid(2)`, `waitpid(2)`, `wait3(3C)`, `write(2)`, `writew(2)`, and `fcntl(2)`, when specifying `F_SETLKW` as the command

When cancellation is asynchronous, cancellation can occur before, during, or after the execution of the function defined as the cancellation point. When cancellation is deferred (the default case), cancellation occurs before the function defined as the cancellation point executes. See `Cancellation Type` for more information about deferred and asynchronous cancellation.

Choosing where to place cancellation points and understanding how cancellation affects your program depend upon your understanding of both your application and of cancellation mechanics.

Typically, any call that might require a long wait should be a cancellation point. Operations need to check for pending cancellation requests when the operation is about to block indefinitely. This includes threads waiting in `pthread_cond_wait(3T)` and `pthread_cond_timedwait(3T)`, threads waiting for the termination of another thread in `pthread_join(3T)`, and threads blocked on `sigwait(2)`.

A mutex is explicitly *not* a cancellation point and should be held for only the minimal essential time.

Most of the dangers in performing cancellations deal with properly restoring invariants and freeing shared resources. For example, a carelessly canceled thread might leave a mutex in a locked state, leading to a deadlock. Or it

| | |
|---------------------------|--|
| | <p>might leave a region of memory allocated with no way to identify it and therefore no way to free it.</p> |
| Cleanup Handlers | <p>When a thread is canceled, it should release resources and clean up the state that is shared with other threads. So, whenever a thread that might be canceled changes the state of the system or of the program, be sure to push a cleanup handler with <code>pthread_cleanup_push(3T)</code> before the cancellation point.</p> <p>When a thread is canceled, all the currently-stacked cleanup handlers are executed in last-in-first-out (LIFO) order. Each handler is run in the scope in which it was pushed. When the last cleanup handler returns, the thread-specific data destructor functions are called. Thread execution terminates when the last destructor function returns.</p> <p>When, in the normal course of the program, an uncanceled thread restores state that it had previously changed, be sure to pop the cleanup handler (that you had set up where the change took place) using <code>pthread_cleanup_pop(3T)</code>. That way, if the thread is canceled later, only currently-changed state will be restored by the handlers that are left in the stack.</p> <p>Be sure to pop the handler in the same scope in which it was pushed. Also, make sure that each push statement has a matching pop statement, or compiler errors will be generated.</p> |
| Cancellation State | <p>Most programmers will use only the default cancellation state of <code>PTHREAD_CANCEL_ENABLE</code>, but can choose to change the state by using <code>pthread_setcancelstate(3T)</code>, which determines whether a thread is cancelable at all. With the default <i>state</i> of <code>PTHREAD_CANCEL_ENABLE</code>, cancellation is enabled, and the thread is cancelable at points determined by its cancellation <i>type</i>. See <i>Cancellation Type</i>.</p> <p>If the <i>state</i> is <code>PTHREAD_CANCEL_DISABLE</code>, cancellation is disabled, and the thread is not cancelable at any point — all cancellation requests to it are held pending.</p> <p>You might want to disable cancellation before a call to a cancel-unsafe library, restoring the old cancel state when the call returns from the library. See <i>Cancel-Safe</i> for explanations of cancel safety.</p> |
| Cancellation Type | <p>A thread's cancellation <i>type</i> is set with <code>pthread_setcanceltype(3T)</code>, and determines whether the thread can be canceled anywhere in its execution, or only at cancellation points.</p> <p>With the default <i>type</i> of <code>PTHREAD_CANCEL_DEFERRED</code>, the thread is cancelable only at cancellation points, and then only when cancellation is enabled.</p> |

If the `type` is `PTHREAD_CANCEL_ASYNCHRONOUS`, the thread is cancelable at any point in its execution (assuming, of course, that cancellation is enabled). Try to limit regions of asynchronous cancellation to sequences with no external dependencies that could result in dangling resources or unresolved state conditions. Using asynchronous cancellation is discouraged because of the danger involved in trying to guarantee correct cleanup handling at absolutely every point in the program.

| Cancellation Type/State Table | | |
|-------------------------------|--|---|
| Type | State | |
| | Enabled (Default) | Disabled |
| Deferred (Default) | Cancellation occurs when the target thread reaches a cancellation point and a cancel is pending. (Default) | All cancellation requests to the target thread are held pending. |
| Asynchronous | Receipt of a <code>pthread_cancel(3T)</code> call causes immediate cancellation. | All cancellation requests to the target thread are held pending; as soon as cancellation is re-enabled, pending cancellations are executed immediately. |

Cancel-Safe

With the arrival of POSIX cancellation, the *cancel-safe* level has been added to the list of MT-Safety levels See `Intro(3)`. An application or library is cancel-safe whenever it has arranged for cleanup handlers to restore system or program state wherever cancellation can occur. The application or library is specifically *Deferred-cancel-safe* when it is cancel-safe for threads whose cancellation type is `PTHREAD_CANCEL_DEFERRED` See Cancellation State. It is specifically *Asynchronous-cancel-safe* when it is cancel-safe for threads whose cancellation type is `PTHREAD_CANCEL_ASYNCHRONOUS`.

Obviously, it is easier to arrange for deferred cancel safety, as this requires system and program state protection only around cancellation points. In general, expect that most applications and libraries are *not* Asynchronous-cancel-safe.

POSIX Threads Only

Note: The cancellation functions described in this reference page are available for POSIX threads, only (the Solaris threads interfaces do not provide cancellation functions).

EXAMPLES

EXAMPLE 1 The following short C++ example shows the pushing/popping of cancellation handlers, the disabling/enabling of cancellation, the use of `pthread_testcancel()`, and so on. The `free_res()` cancellation handler in this example is a dummy function that simply prints a message, but that would free resources in a real application. The function `f2()` is called from the main thread, and goes deep into its call stack by calling itself recursively.

Before `f2()` starts running, the newly created thread has probably posted a cancellation on the main thread since the main thread calls `thr_yield()` right after creating `thread2`. Because cancellation was initially disabled in the main thread, through a call to `pthread_setcancelstate()`, the call to `f2()` from `main()` continues and constructs `X` at each recursive call, even though the main thread has a pending cancellation.

When `f2()` is called for the fifty-first time (when `"i == 50"`), `f2()` enables cancellation by calling `pthread_setcancelstate()`. It then establishes a cancellation point for itself by calling `pthread_testcancel()`. (Because a cancellation is pending, a call to a cancellation point such as `read(2)` or `write(2)` would also cancel the caller here.)

After the `main()` thread is canceled at the fifty-first iteration, all the cleanup handlers that were pushed are called in sequence; this is indicated by the calls to `free_res()` and the calls to the destructor for `X`. At each level, the C++ runtime calls the destructor for `X` and then the cancellation handler, `free_res()`. The print messages from `free_res()` and `X`'s destructor show the sequence of calls.

At the end, the main thread is joined by `thread2`. Because the main thread was canceled, its return status from `pthread_join()` is `PTHREAD_CANCELED`. After the status is printed, `thread2` returns, killing the process (since it is the last thread in the process).

```
#include <pthread.h>
#include <sched.h>
extern "C" void thr_yield(void);

extern "C" void printf(...);

struct X {
    int x;
    X(int i){x = i; printf("X(%d) constructed.\n", i);}
    ~X(){ printf("X(%d) destroyed.\n", x);}
};

void
free_res(void *i)
{
    printf("Freeing `%d`\n", i);
}

char* f2(int i)
{
```

```

    try {
    X dummy(i);
    pthread_cleanup_push(free_res, (void *)i);
    if (i == 50) {
        pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
        pthread_testcancel();
    }
    f2(i+1);
    pthread_cleanup_pop(0);
    }
    catch (int) {
    printf("Error: In handler.\n");
    }
    return "f2";
}

void *
thread2(void *tid)
{
    void *sts;

    printf("I am new thread :%d\n", pthread_self());

    pthread_cancel((pthread_t)tid);

    pthread_join((pthread_t)tid, &sts);

    printf("main thread cancelled due to %d\n", sts);

    return (sts);
}

main()
{
    pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, NULL);
    pthread_create(NULL, NULL, thread2, (void *)pthread_self());
    thr_yield();
    printf("Returned from %s\n",f2(0));
}

```

ATTRIBUTES

See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

[read\(2\)](#), [sigwait\(2\)](#), [write\(2\)](#), [Intro\(3\)](#), [condition\(3T\)](#), [pthread_cleanup_pop\(3T\)](#), [pthread_cleanup_push\(3T\)](#), [pthread_exit\(3T\)](#), [pthread_join\(3T\)](#), [pthread_setcancelstate\(3T\)](#), [pthread_setcanceltype\(3T\)](#), [pthread_testcancel\(3T\)](#), [setjmp\(3C\)](#), [attributes\(5\)](#), [standards\(5\)](#)

| | | | | | | | | | | | | | | | | | | | |
|---------------------|--|---------------------|---|-----------------|---|-----------------|---|-----------------|--|-----------------|--------------------------------|--------------------|---|------------------|---|------------------|---|---------------------|-----------------------------|
| NAME | can_change_color, color_content, COLOR_PAIR, has_colors, init_color, init_pair, pair_content, PAIR_NUMBER, start_color – manipulate color information | | | | | | | | | | | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> bool can_change_color(void); int color_content(short color, short * r, short * g, short * b); int COLOR_PAIR(int n); bool has_colors(void); int init_color(short color, short r, short g, short b); int init_pair(short pair, short fg, short bg); int pair_content(short pair, short * fg, short * bg); int PAIR_NUMBER(int value); int start_color(void);</pre> | | | | | | | | | | | | | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="padding-right: 10px;"><i>color</i></td> <td>Is the number of the color for which to provide information (0 to COLORS).</td> </tr> <tr> <td style="padding-right: 10px;"><i>r</i></td> <td>Is a pointer to the RGB value for the amount of red in <i>color</i> .</td> </tr> <tr> <td style="padding-right: 10px;"><i>g</i></td> <td>Is a pointer to the RGB value for the amount of green in <i>color</i> .</td> </tr> <tr> <td style="padding-right: 10px;"><i>b</i></td> <td>Is a pointer to the RGB value for the amount of blue in <i>color</i> .</td> </tr> <tr> <td style="padding-right: 10px;"><i>n</i></td> <td>Is the number of a color pair.</td> </tr> <tr> <td style="padding-right: 10px;"><i>pair</i></td> <td>Is the number of the color pair for which to provide information (1 to COLOR_PAIRS).</td> </tr> <tr> <td style="padding-right: 10px;"><i>fg</i></td> <td>Is a pointer to the number of the foreground color (0 to COLORS)in <i>pair</i> .</td> </tr> <tr> <td style="padding-right: 10px;"><i>bg</i></td> <td>Is a pointer to the number of the background color (0 to COLORS)in <i>pair</i> .</td> </tr> <tr> <td style="padding-right: 10px;"><i>value</i></td> <td>Is a color attribute value.</td> </tr> </table> | <i>color</i> | Is the number of the color for which to provide information (0 to COLORS). | <i>r</i> | Is a pointer to the RGB value for the amount of red in <i>color</i> . | <i>g</i> | Is a pointer to the RGB value for the amount of green in <i>color</i> . | <i>b</i> | Is a pointer to the RGB value for the amount of blue in <i>color</i> . | <i>n</i> | Is the number of a color pair. | <i>pair</i> | Is the number of the color pair for which to provide information (1 to COLOR_PAIRS). | <i>fg</i> | Is a pointer to the number of the foreground color (0 to COLORS)in <i>pair</i> . | <i>bg</i> | Is a pointer to the number of the background color (0 to COLORS)in <i>pair</i> . | <i>value</i> | Is a color attribute value. |
| <i>color</i> | Is the number of the color for which to provide information (0 to COLORS). | | | | | | | | | | | | | | | | | | |
| <i>r</i> | Is a pointer to the RGB value for the amount of red in <i>color</i> . | | | | | | | | | | | | | | | | | | |
| <i>g</i> | Is a pointer to the RGB value for the amount of green in <i>color</i> . | | | | | | | | | | | | | | | | | | |
| <i>b</i> | Is a pointer to the RGB value for the amount of blue in <i>color</i> . | | | | | | | | | | | | | | | | | | |
| <i>n</i> | Is the number of a color pair. | | | | | | | | | | | | | | | | | | |
| <i>pair</i> | Is the number of the color pair for which to provide information (1 to COLOR_PAIRS). | | | | | | | | | | | | | | | | | | |
| <i>fg</i> | Is a pointer to the number of the foreground color (0 to COLORS)in <i>pair</i> . | | | | | | | | | | | | | | | | | | |
| <i>bg</i> | Is a pointer to the number of the background color (0 to COLORS)in <i>pair</i> . | | | | | | | | | | | | | | | | | | |
| <i>value</i> | Is a color attribute value. | | | | | | | | | | | | | | | | | | |

DESCRIPTION

The **start_color()** function initializes the use of color. It must be used if color is to be used in the program. It must be called before any other color functions, ideally right after **initscr(3XC)**. Eight basic colors are initialized (black, red, green, yellow, blue, magenta, cyan, and white) and two global variables (**COLORS** and **COLOR_PAIRS**). The former variable specifies the number of colors the terminal supports, the latter the number of color pairs. Colors are always in pairs consisting of a foreground color (for characters) and a background color (for the the rest of the character cell). The initial appearance of these colors is unspecified.

The **init_pair()** function initializes a color pair so that it can be used as a parameter. **COLOR_PAIR()** can be used as an attribute and as a parameter to functions like **attr_set(3XC)**. Its first parameter is the number of the color pair to be changed; the second parameter is the number of the foreground color; the third parameter is the number of the background color. The maximum number of color pairs and colors the terminal can support are defined in the global variables **COLOR_PAIRS** and **COLORS**, respectively.

Color pair 0 (zero) is reserved for use by X/Open Curses.

Each time that a color pair is initialized, the screen is refreshed and all occurrences of that color pair are updated to reflect the new definition.

The **init_color()** function redefines the color using the number of the color and the RGB values for red, green, and blue as parameters.

The following default colors are defined (X/Open Curses assumes that **COLOR_BLACK** is the default background color for all terminals):

```
COLOR_BLACK
COLOR_RED
COLOR_GREEN
COLOR_YELLOW
COLOR_BLUE
COLOR_MAGENTA
COLOR_CYAN
COLOR_WHITE
```

Each time that a color is redefined with the **init_color()** function, the screen is refreshed and all occurrences of that color are updated to reflect the new definition.

The **can_change_color()** function returns **TRUE** if the terminal supports color and the colors can be changed. The **has_colors()** function returns **TRUE** if the terminal supports color. These functions are useful when writing

terminal-independent programs. They can be used to determine whether to replace color with another attribute on a particular terminal.

The **color_content()** function provides information on the amount of red, green, and blue in a particular color. The intensity of each color is stored in the addresses pointed to by the *r*, *g*, and *b* parameters, respectively. The values passed back range from 0 (zero) (no component of that color) to 1000 (maximum amount of component).

The **pair_content()** function provides information on what colors compose the specified color pair. The numbers of the foreground and background colors are passed back in the addresses pointed to by the *fg* and *bg* parameters, respectively. The values stored in *fg* and *bg* range from 0 (zero) to COLORS .

RETURN VALUES

The **has_colors()** function returns TRUE if the terminal is able to handle colors. Otherwise, it returns FALSE .

The **can_change_color()** function returns TRUE if the terminal supports colors and is able to change their definitions. Otherwise, it returns FALSE .

On success, the other functions return OK . Otherwise, they return ERR .

ERRORS

None.

SEE ALSO

attroff(3XC) , **delscreen(3XC)**

| NAME | catgets – read a program message | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <nl_types.h> char *catgets(nl_catd catd, int set_num, int msg_num, const char *s);</pre> | | | | |
| DESCRIPTION | The catgets() function attempts to read message <i>msg_num</i> , in set <i>set_num</i> , from the message catalog identified by <i>catd</i> . The <i>catd</i> argument is a catalog descriptor returned from an earlier call to catopen() . The <i>s</i> argument points to a default message string which will be returned by catgets() if the identified message catalog is not currently available. | | | | |
| RETURN VALUES | If the identified message is retrieved successfully, catgets() returns a pointer to an internal buffer area containing the null terminated message string. If the call is unsuccessful for any reason, catgets() returns a pointer to <i>s</i> and <i>errno</i> may be set to indicate the error. | | | | |
| ERRORS | <p>The catgets() function may fail if:</p> <p>EBADF The <i>catd</i> argument is not a valid message catalogue descriptor open for reading.</p> <p>EINTR The read operation was terminated due to the receipt of a signal, and no data was transferred.</p> <p>EINVAL The message catalog identified by <i>catd</i> is corrupted.</p> <p>ENOMSG The message identified by <i>set_id</i> and <i>msg_id</i> is not in the message catalog.</p> | | | | |
| USAGE | The catgets() function can be used safely in multithreaded applications, as long as setlocale(3C) is not being called to change the locale. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | <p>gencat(1), catclose(3C), catopen(3C), gettext(3C), setlocale(3C), attributes(5)</p> <p><i>Solaris Internationalization Guide For Developers</i></p> | | | | |

| | |
|--------------------|--|
| NAME | catopen, catclose – open/close a message catalog |
| SYNOPSIS | <pre>#include <nl_types.h> nl_catd catopen(const char * name, int oflag); int catclose(nl_catd catd);</pre> |
| DESCRIPTION | <p>The catopen() function opens a message catalog and returns a message catalog descriptor. <i>name</i> specifies the name of the message catalog to be opened. If <i>name</i> contains a “/”, then <i>name</i> specifies a complete pathname for the message catalog; otherwise, the environment variable NLSPATH is used and /usr/lib/locale/ locale / LC_MESSAGES must exist. If NLSPATH does not exist in the environment, or if a message catalog cannot be opened in any of the paths specified by NLSPATH , then the default path /usr/lib/locale/ locale / LC_MESSAGES is used. In the "C" locale, catopen() will always succeed without checking the default search path.</p> <p>The names of message catalogs and their location in the filesystem can vary from one system to another. Individual applications can choose to name or locate message catalogs according to their own special needs. A mechanism is therefore required to specify where the catalog resides.</p> <p>The NLSPATH variable provides both the location of message catalogs, in the form of a search path, and the naming conventions associated with message catalog files. For example:</p> <pre>NLSPATH =/nlslib/%L/%N.cat:/nlslib/%N/%L</pre> <p>The metacharacter % introduces a substitution field, where %L substitutes the current setting of either the LANG environment variable, if the value of <i>oflag</i> is 0 , or the LC_MESSAGES category, if the value of <i>oflag</i> is NL_CAT_LOCALE , and %N substitutes the value of the <i>name</i> parameter passed to catopen() . Thus, in the above example, catopen() will search in /nlslib/\$ LANG / <i>name</i> .cat, if <i>oflag</i> is 0 , or in /nlslib/{ LC_MESSAGES }/ <i>name</i> .cat , if <i>oflag</i> is NL_CAT_LOCALE .</p> <p>NLSPATH will normally be set up on a system wide basis (in /etc/profile)and thus makes the location and naming conventions associated with message catalogs transparent to both programs and users.</p> <p>The full set of metacharacters is:</p> <ul style="list-style-type: none"> %N The value of the name parameter passed to catopen() . %L The value of LANG or LC_MESSAGES . %l The value of the <i>language</i> element of LANG or LC_MESSAGES . |

`%t` The value of the *territory* element of LANG or LC_MESSAGES .

`%c` The value of the *codeset* element of LANG or LC_MESSAGES .

`%%` A single %.

The LANG environment variable provides the ability to specify the user's requirements for native languages, local customs and character set, as an ASCII string in the form

```
LANG =language[_territory[.codeset]]
```

A user who speaks German as it is spoken in Austria and has a terminal which operates in ISO 8859/1 codeset, would want the setting of the LANG variable to be

```
LANG =De_A.88591
```

With this setting it should be possible for that user to find any relevant catalogs should they exist.

Should the LANG variable not be set, the value of LC_MESSAGES as returned by **setlocale()** is used. If this is `NULL`, the default path as defined in **nl_types()** is used.

A message catalogue descriptor remains valid in a process until that process closes it, or a successful call to one of the `exec` functions. A change in the setting of the LC_MESSAGES category may invalidate existing open catalogues.

If a file descriptor is used to implement message catalogue descriptors, the `FD_CLOEXEC` flag will be set; see `<fcntl.h>` .

If the value of *oflag* argument is 0 , the LANG environment variable is used to locate the catalogue without regard to the LC_MESSAGES category. If the *oflag* argument is `NL_CAT_LOCALE` , the LC_MESSAGES category is used to locate the message catalogue.

The **catclose()** function closes the message catalog identified by *catd* . If a file descriptor is used to implement the type `nl_catd` , that file descriptor will be closed.

RETURN VALUES

Upon successful completion, **catopen()** returns a message catalog descriptor for use on subsequent calls to **catgets()** and **catclose()** . Otherwise it returns `(nl_catd) -1` .

Upon successful completion, **catclose()** returns 0 . Otherwise it returns -1 and sets **errno** to indicate the error.

ERRORS

The **catopen()** function may fail if:

EACCES Search permission is denied for the component of the path prefix of the message catalogue or read permission is denied for the message catalogue.

EMFILE There are **OPEN_MAX** file descriptors currently open in the calling process.

ENAMETOOLONG The length of the pathname of the message catalogue exceeds **PATH_MAX** , or a pathname component is longer than **NAME_MAX** .

ENAMETOOLONG Pathname resolution of a symbolic link produced an intermediate result whose length exceeds **PATH_MAX** .

ENFILE Too many files are currently open in the system.

ENOENT The message catalogue does not exist or the *name* argument points to an empty string.

ENOMEM Insufficient storage space is available.

ENOTDIR A component of the path prefix of the message catalogue is not a directory.

The **catclose()** function may fail if:

EBADF The catalogue descriptor is not valid.

EINTR The **catclose()** function was interrupted by a signal.

USAGE

The **catopen()** and **catclose()** functions can be used safely in multithreaded applications, as long as **setlocale(3C)** is not being called to change the locale.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

gencat(1) , **catgets(3C)** , **gettext(3C)** , **setlocale(3C)** , **attributes(5)** , **environ(5)** , **nl_types(5)**

| | |
|----------------------|---|
| NAME | cbreak, nocbreak, noraw, raw – set input mode controls |
| SYNOPSIS | <pre>#include <curses.h> int cbreak(void); int nocbreak(void); int noraw(void); int raw(void);</pre> |
| DESCRIPTION | <p>The cbreak() function enables the character input mode. This overrides any previous call to the raw() function and turns the <code>stty</code> flag <code>ICANON</code> off.</p> <p>The nocbreak() function sets the line canonical mode and turns the <code>stty</code> flag <code>ICANON</code> on without touching the <code>ISIG</code> or <code>IXON</code> flags.</p> <p>The noraw() function sets the line canonical mode and turns the the <code>stty</code> flags <code>ICANON</code> , <code>ISIG</code> , and <code>IXON</code> all on.</p> <p>The raw() function sets the character input mode and turns the <code>stty</code> flags <code>ICANON</code> , <code>ISIG</code> , and <code>IXON</code> all off. This mode provides maximum control over input.</p> <p>It is important to remember that the terminal may or may not be in character mode operation initially. Most interactive programs require cbreak() to be enabled.</p> |
| RETURN VALUES | On success, these functions return <code>OK</code> . Otherwise, they return <code>ERR</code> . |
| ERRORS | None. |
| SEE ALSO | <code>getch(3XC)</code> , <code>halfdelay(3XC)</code> , <code>nodelay(3XC)</code> , <code>timeout(3XC)</code> , <code>termio(7I)</code> |

NAME | cbrt – cube root function

SYNOPSIS | cc [*flag ...*] *file ...* -lm [*library ...*]
 | #include <math.h>
 | double **cbrt**(double *x*);

DESCRIPTION | The **cbrt()** function computes the cube root of *x*.

RETURN VALUES | On successful completion, **cbrt()** returns the cube root of *x*. If *x* is NaN, **cbrt()** returns NaN.

ERRORS | No errors will occur.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **attributes(5)**

NAME | ceil – ceiling value function

SYNOPSIS | cc [*flag ...*] *file ...* -lm [*library ...*]
 | #include <math.h>
 | double **ceil**(double *x*);

DESCRIPTION | The **ceil()** function computes the smallest integral value not less than *x*.

RETURN VALUES | Upon successful completion, **ceil()** returns the smallest integral value not less than *x*, expressed as a type `double`.
 | If *x* is NaN, NaN is returned.
 | If *x* is ±Inf or ±0, *x* is returned.

ERRORS | No errors will occur.

USAGE | The integral value returned by **ceil()** as a `double` may not be expressible as an `int` or `long int`. The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **floor(3M)**, **isnan(3M)**, **attributes(5)**

NAME cfgetispeed, cfgetospeed – get input and output baud rate

SYNOPSIS #include <termios.h>

```
speed_t cfgetispeed(const struct termios * termios_p);
speed_t cfgetospeed(const struct termios * termios_p);
```

DESCRIPTION The **cfgetispeed()** function extracts the input baud rate from the `termios` structure to which the `termios_p` argument points.

The **cfgetospeed()** function extracts the output baud rate from the `termios` structure to which the `termios_p` argument points.

These functions returns exactly the value in the `termios` data structure, without interpretation.

RETURN VALUES Upon successful completion, **cfgetispeed()** returns a value of type `speed_t` representing the input baud rate.

Upon successful completion, **cfgetospeed()** returns a value of type `speed_t` representing the output baud rate.

ERRORS No errors are defined.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|--------------------------------|
| MT-Level | MT-Safe, and Async-Signal-Safe |

SEE ALSO **cfgetospeed(3)**, **tcgetattr(3)**, **attributes(5)**, **termio(7I)**

| NAME | cfsetispeed, cfsetospeed – set input and output baud rate | | | | |
|----------------------|--|----------------|-----------------|----------|--------------------------------|
| SYNOPSIS | <pre>#include <termios.h> int cfsetispeed(struct termios * <i>termios_p</i>, speed_t <i>speed</i>); int cfsetospeed(struct termios * <i>termios_p</i>, speed_t <i>speed</i>);</pre> | | | | |
| DESCRIPTION | <p>The cfsetispeed() function sets the input baud rate stored in the structure pointed to by <i>termios_p</i> to <i>speed</i>.</p> <p>The cfsetospeed() function sets the output baud rate stored in the structure pointed to by <i>termios_p</i> to <i>speed</i>.</p> <p>There is no effect on the baud rates set in the hardware until a subsequent successful call to tcsetattr(3) on the same <i>termios</i> structure.</p> | | | | |
| RETURN VALUES | Upon successful completion, cfsetispeed() and cfsetospeed() return 0 . Otherwise -1 is returned, and <i>errno</i> may be set to indicate the error. | | | | |
| ERRORS | <p>The cfsetispeed() and cfsetospeed() functions may fail if:</p> <p>EINVAL The <i>speed</i> value is not a valid baud rate.</p> <p>EINVAL The value of <i>speed</i> is outside the range of possible speed values as specified in <i><termios.h></i> .</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe, and Async-Signal-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe, and Async-Signal-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe, and Async-Signal-Safe | | | | |
| SEE ALSO | cfgetispeed(3) , tcsetattr(3) , attributes(5) , termio(7I) | | | | |

| | |
|--------------------|---|
| NAME | chgat, mvchgat, mvwchgat, wchgat – change the rendition of characters in a window |
| SYNOPSIS | <pre>#include <curses.h> int chgat(int n, attr_t attr, short color, void *const opts); int mvchgat(int y, int x, int n, attr_t attr, short color, void *const opts); int mvwchgat(WINDOW * win, int y, int x, int n, attr_t attr, short color, void *const opts); int wchgat(WINDOW * win, int n, attr_t attr, short color, void *const opts);</pre> |
| PARAMETERS | <p><i>n</i> Is the number of characters whose rendition is to be changed.</p> <p><i>attr</i> Is the set of attributes to be assigned to the characters.</p> <p><i>color</i> Is the new color pair to be assigned to the characters.</p> <p><i>opts</i> Is reserved for future use. Currently, this must be a null pointer.</p> <p><i>y</i> Is the y (row) coordinate of the starting position in the window.</p> <p><i>x</i> Is the x (column) coordinate of the starting position in the window. changed in the window.</p> <p><i>win</i> Is a pointer to the window in which the rendition of characters is to be changed.</p> |
| DESCRIPTION | <p>The chgat() and wchgat() functions change the rendition (that is, the attributes and color pair) associated with the next <i>n</i> characters beginning at the current cursor position in the windows <i>stdscr</i> and <i>win</i> , respectively. The mvchgat() and mvwchgat() perform identical actions but beginning with the position indicated by <i>x</i> (column) and <i>y</i> (row) instead of the current cursor position. If <i>n</i> is less than 0, these functions change the rendition of all characters from the starting position to the end of that line. The cursor position is not changed.</p> |
| ERRORS | <p>OK Successful completion.</p> <p>ERR An error occurred.</p> |

chgat(3XC)

X/Open Curses Library Functions

SEE ALSO | `bkgrnd(3XC)` , `setcchar(3XC)`

NAME cldap_close – dispose of connectionless LDAP pointer

SYNOPSIS cc[*flag...*] *file...* -ldap[*library...*]

```
#include <lber.h>
#include <ldap.h>
```

```
void cldap_close(LDAP *ld);
```

PARAMETERS

ld The LDAP pointer returned by a previous call to `clldap_open(3N)`.

DESCRIPTION

The `clldap_close()` function disposes of memory allocated by `clldap_open(3N)`. It should be called when all CLDAP communication is complete.

ATTRIBUTES

See `attributes(5)` for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|--|
| Availability | SUNWlldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

`ldap(3N)`, `clldap_open(3N)`, `clldap_search_s(3N)`, `clldap_setretryinfo(3N)`

| NAME | cldap_open – LDAP connectionless communication preparation | | | | | | |
|--------------------|--|----------------|-----------------|--------------|--|-----------------|----------|
| SYNOPSIS | <pre>cc[<i>flag...</i>] <i>file...</i> -lldap[<i>library...</i>] #include <lber.h> #include <ldap.h> LDAP *cldap_open(char *host, int port);</pre> | | | | | | |
| PARAMETERS | <p>host The name of the host on which the LDAP server is running.</p> <p>port The port number to connect.</p> | | | | | | |
| DESCRIPTION | <p>The cldap_open() function is called to prepare for connectionless LDAP communication (over udp(7P)). It allocates an LDAP structure which is passed to future search requests.</p> <p>If the default IANA-assigned port of 389 is desired, LDAP_PORT should be specified for <i>port</i>. <i>host</i> can contain a space-separated list of hosts or addresses to try. cldap_open() returns a pointer to an LDAP structure, which should be passed to subsequent calls to cldap_search_s(3N), cldap_setretryinfo(3N), and cldap_close(3N). Certain fields in the LDAP structure can be set to indicate size limit, time limit, and how aliases are handled during operations. See ldap_open(3N) and <ldap.h> for more details.</p> | | | | | | |
| ERRORS | If an error occurs, cldap_open() will return NULL and errno will be set appropriately. | | | | | | |
| ATTRIBUTES | See attributes(5) for a description of the following attributes: | | | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>Availability</td> <td>SUNWlldap (32-bit) SUNWldapx (64-bit)</td> </tr> <tr> <td>Stability Level</td> <td>Evolving</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | Availability | SUNWlldap (32-bit) SUNWldapx (64-bit) | Stability Level | Evolving |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| Availability | SUNWlldap (32-bit) SUNWldapx (64-bit) | | | | | | |
| Stability Level | Evolving | | | | | | |
| SEE ALSO | ldap(3N) , cldap_search_s(3N) , cldap_setretryinfo(3N) , cldap_close(3N) , udp(7P) | | | | | | |

| | |
|---------------------------------|---|
| NAME | cldap_search_s – connectionless LDAP search |
| SYNOPSIS | <pre>cc[flag...] file... -ldap[library...] #include <lber.h> #include <ldap.h> int cldap_search_s(LDAP *ld, char *base, int scope, char *filter, char *attrs, int attrsonly, LDAPMessage **res, char *logdn);</pre> |
| DESCRIPTION | <p>The cldap_search_s() function performs an LDAP search using the Connectionless LDAP (CLDAP) protocol.</p> <p>cldap_search_s() has parameters and behavior identical to that of ldap_search_s(3N), except for the addition of the <i>logdn</i> parameter. <i>logdn</i> should contain a distinguished name to be used only for logging purposed by the LDAP server. It should be in the text format described by RFC 1779 <i>A String Representation of Distinguished Names</i>.</p> |
| Retransmission Algorithm | <p>cldap_search_s() operates using the CLDAP protocol over udp(7P). Since UDP is a non-reliable protocol, a retry mechanism is used to increase reliability. The ldap_setretryinfo(3N) function can be used to set two retry parameters: <i>tries</i>, a count of the number of times to send a search request and <i>timeout</i>, an initial timeout that determines how long to wait for a response before re-trying. <i>timeout</i> is specified seconds. These values are stored in the <code>ld_cldaptries</code> and <code>ld_cldaptimeout</code> members of the <code>ld</code> LDAP structure, and the default values set in ldap_open(3N) are 4 and 3 respectively. The retransmission algorithm used is:</p> <p>Step 1. Set the current timeout to <code>ld_cldaptimeout</code> seconds, and the current LDAP server address to the first LDAP server found during the ldap_open(3N) call.</p> <p>Step 2: Send the search request to the current LDAP server address.</p> <p>Step 3: Set the wait timeout to the current timeout divided by the number of server addresses found during ldap_open(3N) or to one second, whichever is larger. Wait at most that long for a response; if a response is received, STOP. Note that the wait timeout is always rounded down to the next lowest second.</p> <p>Step 5: Repeat steps 2 and 3 for each LDAP server address.</p> <p>Step 6: Set the current timeout to twice its previous value and repeat Steps 2 through 6 a maximum of <i>tries</i> times.</p> |

EXAMPLES

Assume that the default values for *tries* and *timeout* of 4 tries and 3 seconds are used. Further, assume that a space-separated list of two hosts, each with one address, was passed to `cldap_open(3N)`. The pattern of requests sent will be (stopping as soon as a response is received):

```
Time  Search Request Sent To:
+0   Host A try 1
+1   (0+3/2) Host B try 1
+2   (1+3/2) Host A try 2
+5   (2+6/2) Host B try 2
+8   (5+6/2) Host A try 3
+14  (8+12/2) Host B try 3
+20  (14+12/2) Host A try 4
+32  (20+24/2) Host B try 4
+44  (20+24/2) (give up - no response)
```

ERRORS

`cldap_search_s()` returns `LDAP_SUCCESS` if a search was successful and the appropriate LDAP error code otherwise. See `ldap_error(3N)` for more information.

ATTRIBUTES

See `attributes(5)` for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|--|
| Availability | SUNWlldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

`ldap(3N)`, `ldap_error(3N)`, `ldap_search_s(3N)`, `cldap_open(3N)`, `cldap_setretryinfo(3N)`, `cldap_close(3N)`, `udp(7P)`

NAME cldap_setretryinfo – set connectionless LDAP request retransmission parameters

SYNOPSIS cc[*flag...*] *file...* -ldap[*library...*]

```
#include <lber.h>
#include <ldap.h>

void cldap_setretryinfo(LDAP *ld, int tries, int timeout);
```

PARAMETERS

ld LDAP pointer returned from a previous call to `cldap_open(3N)`.

tries Maximum number of times to send a request.

timeout Initial time, in seconds, to wait before re-sending a request.

DESCRIPTION The `cldap_setretryinfo()` function is used to set the CLDAP request retransmission behavior for future `cldap_search_s(3N)` calls. The default values (set by `cldap_open(3N)`) are 4 tries and 3 seconds between tries. See `cldap_search_s(3N)` for a complete description of the retransmission algorithm used.

ATTRIBUTES See `attributes(5)` for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|---|
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO `ldap(3N)`, `cldap_open(3N)`, `cldap_search_s(3N)`, `cldap_close(3N)`

| | |
|--------------------|---|
| NAME | clear, erase, wclear, werase – clear a window |
| SYNOPSIS | <pre>#include <curses.h> int clear(void); int erase(void); int wclear(WINDOW * win); int werase(WINDOW * win);</pre> |
| PARAMETERS | <p>win Is a pointer to the window that is to be cleared.</p> |
| DESCRIPTION | <p>The clear() and erase() functions clear <code>stdscr</code>, destroying its previous contents. The wclear() and werase() functions perform the same action, but clear the window specified by <i>win</i> instead of <code>stdscr</code>.</p> <p>The clear() and wclear() functions also call the clearok() function. This function clears and redraws the entire screen on the next call to refresh(3XC) or wrefresh(3XC) for the window.</p> <p>The current background character (and attributes) is used to clear the screen.</p> |
| ERRORS | <p>OK Successful completion.</p> <p>ERR An error occurred.</p> |
| SEE ALSO | <p>bkgdset(3XC), clearok(3XC), clrtoeol(3XC), clrtoeol(3XC), doupdate(3XC), refresh(3XC), wrefresh(3XC)</p> |

| | |
|--------------------|--|
| NAME | clearok, idlok, leaveok, scrollok, setscrreg, wsetscrreg – set terminal output controls |
| SYNOPSIS | <pre>#include <curses.h> int clearok(WINDOW * win, bool bf); int idlok(WINDOW * win, bool bf); int leaveok(WINDOW * win, bool bf); int scrollok(WINDOW * win, bool bf); int setscrreg(int top, int bot); int wsetscrreg(WINDOW * win, int top, int bot);</pre> |
| PARAMETERS | <p>win Is a pointer to a window.</p> <p>bf Is a Boolean expression.</p> <p>top Is the top line of the scrolling region (top of the window is line 0).</p> <p>bot Is the bottom line of the scrolling region (top of the window is line 0).</p> |
| DESCRIPTION | <p>These functions set options that deal with the output of X/Open Curses functions. The clearok() function checks the value of the Boolean expression <i>bf</i> . If <i>bf</i> is TRUE , clearok() clears and redraws the entire screen on the next call to refresh(3XC) . If <i>win</i> is <code>curscr</code> , the next call to refresh() for <i>any</i> window clears and redraws the screen.</p> <p>The idlok() function enables (<i>bf</i> is TRUE)or disables (<i>bf</i> is FALSE)the use of the insert/delete line capability of the terminal, provided that the terminal supports the operation. By default, the use of insert/delete line is disabled because its use is undesirable for most applications (screen editor applications are one exception). When disabled, X/Open Curses redraws the changed portions of all lines.</p> <p>The leaveok() function controls the cursor positioning following a call to the refresh() function. If <i>bf</i> is TRUE , leaveok() leaves the cursor in a position that X/Open Curses finds convenient at the time that the window is refreshed. Normally, when a window is refreshed, leaveok() is disabled and the cursor is mapped from the logical window to the same location on the physical screen.</p> <p>Enabling leaveok() is useful when the cursor is not used or is not important in the application. Reducing cursor movements simplifies program interaction.</p> |

Once **leaveok()** is set to `TRUE` , it remains enabled until another call sets it to `FALSE` , or until the program terminates.

The **scrollok()** function controls what happens when the cursor advances outside the scrolling region. When enabled, if the cursor advances outside the scrolling region or a call to the `scr1(3XC)` function is made, the screen scrolls up one line.

The terminal screen will produce a scrolling effect if **idlok()** is also enabled.

The **setscreg()** and **wsetscreg()** functions set up scrolling regions in the windows `stdscr` and `win` , respectively. The dimensions of the scrolling region are defined by the *top* and *bottom* parameter. If **scrollok()** is enabled and the cursor is on the last line of the scroll region, any attempt to move the cursor beyond the bottom margin of the scrolling region scrolls the scrolling region up by one line. By default, the scrolling region of a window is the entire window.

For full screen windows, the terminal screen produces a scrolling effect if **idlok()** is also enabled.

RETURN VALUES

On success, the **setscreg()** and **wsetscreg()** functions return `OK` . Otherwise, they return `ERR` .

The other functions always return `OK` .

ERRORS

None.

SEE ALSO

`bkgdset(3XC)` , `clear(3XC)` , `doupdate(3XC)` , `scr1(3XC)`

| NAME | clock – report CPU time used | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | #include <time.h> clock_t clock(void); | | | | |
| DESCRIPTION | The clock() function returns the amount of CPU time (in microseconds) used since the first call to clock() in the calling process. The time reported is the sum of the user and system times of the calling process and its terminated child processes for which it has executed the wait(2) function, the pclose(3S) function, or the system(3S) function. | | | | |
| RETURN VALUES | Dividing the value returned by clock() by the constant CLOCKS_PER_SEC , defined in the <time.h> header, will give the time in seconds. If the process time used is not available or cannot be represented, clock returns the value (clock_t) -1. | | | | |
| USAGE | The value returned by clock() is defined in microseconds for compatibility with systems that have CPU clocks with much higher resolution. Because of this, the value returned will wrap around after accumulating only 2147 seconds of CPU time (about 36 minutes). | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | times(2) , wait(2) , popen(3S) , system(3S) , attributes(5) | | | | |

| | |
|----------------------|---|
| NAME | clock_settime, clock_gettime, clock_getres – high-resolution clock operations |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lrt [<i>library</i> ...] #include <time.h> int clock_settime(clockid_t <i>clock_id</i>, const struct timespec * <i>tp</i>); int clock_gettime(clockid_t <i>clock_id</i>, struct timespec * <i>tp</i>); int clock_getres(clockid_t <i>clock_id</i>, struct timespec * <i>res</i>);</pre> |
| DESCRIPTION | <p>The clock_settime() function sets the specified clock, <i>clock_id</i>, to the value specified by <i>tp</i>. Time values that are between two consecutive non-negative integer multiples of the resolution of the specified clock are truncated down to the smaller multiple of the resolution.</p> <p>The clock_gettime() function returns the current value <i>tp</i> for the specified clock, <i>clock_id</i>.</p> <p>The resolution of any clock can be obtained by calling clock_getres(). Clock resolutions are system-dependent and cannot be set by a process. If the argument <i>res</i> is not NULL, the resolution of the specified clock is stored in the location pointed to by <i>res</i>. If <i>res</i> is NULL, the clock resolution is not returned. If the time argument of clock_settime() is not a multiple of <i>res</i>, then the value is truncated to a multiple of <i>res</i>.</p> <p>A clock may be systemwide (that is, visible to all processes) or per-process (measuring time that is meaningful only within a process). A <i>clock_id</i> of CLOCK_REALTIME is defined in <time.h>. This clock represents the realtime clock for the system. For this clock, the values returned by clock_gettime() and specified by clock_settime() represent the amount of time (in seconds and nanoseconds) since the Epoch. Additional clocks may also be supported. The interpretation of time values for these clocks is unspecified.</p> |
| RETURN VALUES | A return value of 0 indicates that the call succeeded. A return value of -1 indicates that an error occurred, and <i>errno</i> is set to indicate the error. |
| ERRORS | The clock_settime() , clock_gettime() and clock_getres() functions will fail if: EINVAL The <i>clock_id</i> argument does not specify a known clock. |

ENOSYS The functions **clock_settime()** , **clock_gettime()** , and **clock_getres()** are not supported by this implementation.

The **clock_settime()** function will fail if:

EINVAL The *tp* argument to **clock_settime()** is outside the range for the given clock ID; or the *tp* argument specified a nanosecond value less than zero or greater than or equal to 1000 million.

The **clock_settime()** function may fail if:

EPERM The requesting process does not have the appropriate privilege to set the specified clock.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|---------------------------------------|
| MT-Level | clock_gettime() is Async-Signal-Safe |

SEE ALSO

time(2) , **ctime(3C)** , **timer_gettime(3R)** , **attributes(5)** , **time(5)**

NAME closedir – close a directory stream

SYNOPSIS

```
#include <sys/types.h>
#include <dirent.h>

int closedir(DIR *dirp);
```

DESCRIPTION The **closedir()** function closes the directory stream referred to by the argument *dirp*. Upon return, the value of *dirp* may no longer point to an accessible object of the type `DIR`. If a file descriptor is used to implement type `DIR`, that file descriptor will be closed.

RETURN VALUES Upon successful completion, **closedir()** returns 0. Otherwise, -1 is returned and `errno` is set to indicate the error.

ERRORS The **closedir()** function may fail if:

EBADF The *dirp* argument does not refer to an open directory stream.

EINTR The **closedir()** function was interrupted by a signal.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO **opendir(3C)**, **attributes(5)**

| | |
|----------------------|---|
| NAME | clrtoBot, wclrtoBot – clear to the end of a window |
| SYNOPSIS | <pre>#include <curses.h> int clrtoBot(void); int wclrtoBot(WINDOW * win);</pre> |
| PARAMETERS | <p>win Is a pointer to the window that is to be cleared.</p> |
| DESCRIPTION | <p>The clrtoBot() function clears all characters in the <code>stdscr</code> window from the cursor to the end of the window. The wclrtoBot() function performs the same action in the window specified by <i>win</i> instead of in <code>stdscr</code>. The current background character (and rendition) is used to clear the screen.</p> <p>If the clearing action results in clearing only a portion of a multicolumn character, background characters are displayed in place of the remaining portion.</p> |
| RETURN VALUES | On success, these functions return <code>OK</code> . Otherwise, they return <code>ERR</code> . |
| ERRORS | None. |
| SEE ALSO | <code>bkgdset(3XC)</code> , <code>clear(3XC)</code> , <code>clearok(3XC)</code> , <code>crltoeol(3XC)</code> |

| | |
|----------------------|--|
| NAME | clrtoeol, wclrtoeol – clear to the end of a line |
| SYNOPSIS | #include <curses.h> int clrtoeol(void); int wclrtoeol(WINDOW * win); |
| PARAMETERS | win Is a pointer to the window in which to clear to the end of the line. |
| DESCRIPTION | The clrtoeol() function clears the current line from the cursor to the right margin in the <code>stdscr</code> window. The wclrtoeol() function performs the same action, but in the window specified by <i>win</i> instead of <code>stdscr</code> . The current background character (and rendition) is used to clear the screen. If the clearing action results in clearing only a portion of a multicolumn character, background characters are displayed in place of the remaining portion. |
| RETURN VALUES | On success, these functions return <code>OK</code> . Otherwise, they return <code>FALSE</code> . |
| ERRORS | None. |
| SEE ALSO | <code>bkgdset(3XC)</code> , <code>clear(3XC)</code> , <code>clearok(3XC)</code> , <code>clrtoeol(3XC)</code> |

| | |
|--------------------|--|
| NAME | cond_init, cond_wait, cond_timedwait, cond_signal, cond_broadcast, cond_destroy – condition variables |
| SYNOPSIS | <pre>cc - mt [flag ...] file ...[library ...] #include <thread.h> #include <synch.h> int cond_init(cond_t * cvp, int type, void * arg); int cond_wait(cond_t * cvp, mutex_t * mp); int cond_timedwait(cond_t * cvp, mutex_t * mp, timestruc_t * abstime); int cond_signal(cond_t * cvp); int cond_broadcast(cond_t * cvp); int cond_destroy(cond_t * cvp);</pre> |
| DESCRIPTION | |
| Initialize | <p>Condition variables and mutexes should be global. Condition variables that are allocated in writable memory can synchronize threads among processes if they are shared by the cooperating processes (see <code>mmap(2)</code>) and are initialized for this purpose.</p> <p>The scope of a condition variable is either intra-process or inter-process. This is dependent upon whether the argument is passed implicitly or explicitly to the initialization of that condition variable. A condition variable does not need to be explicitly initialized. A condition variable is initialized with all zeros, by default, and its scope is set to within the calling process. For inter-process synchronization, a condition variable must be initialized once, and only once, before use.</p> <p>A condition variable must not be simultaneously initialized by multiple threads or re-initialized while in use by other threads.</p> <p>Condition variables' attributes may be set to the default or customized at initialization.</p> <p><code>cond_init()</code> initializes the condition variable pointed to by <code>cvp</code>. A condition variable can have several different types of behavior, specified by <code>type</code>. No</p> |

current type uses *arg* although a future type may specify additional behavior parameters via *arg*. *type* may be one of the following:

| | |
|---------------|--|
| USYNC_THREAD | The condition variable can synchronize threads only in this process. This is the default. |
| USYNC_PROCESS | The condition variable can synchronize threads in this process and other processes. Only one process should initialize the condition variable. The object initialized with this attribute must be allocated in memory shared between processes, either in System V shared memory (see <code>shmop(2)</code>) or in memory mapped to a file (see <code>mmap(2)</code>). It is illegal to initialize the object this way and to not allocate it in such shared memory. |

Initializing condition variables can also be accomplished by allocating in zeroed memory, in which case, a *type* of USYNC_THREAD is assumed.

If default condition variable attributes are used, statically allocated condition variables can be initialized by the macro DEFAULTTCV.

Default condition variable initialization (intra-process):

```
cond_t  cvp;
cond_init(&cvp, NULL, NULL); /* initialize condition variable with default */
OR
cond_init(&cvp, USYNC_THREAD, NULL);
OR
cond_t  cond = DEFAULTTCV;
```

Customized condition variable initialization (inter-process):

```
cond_init(&cvp, USYNC_PROCESS, NULL); /* initialize cv with inter-process scope */
```

Condition Wait

The condition wait interface allows a thread to wait for a condition and atomically release the associated mutex that it needs to hold to check the condition. The thread waits for another thread to make the condition true and that thread's resulting call to signal and wakeup the waiting thread.

Condition Signaling

cond_wait() atomically releases the mutex pointed to by *mp* and causes the calling thread to block on the condition variable pointed to by *cvp*. The blocked thread may be awakened by **cond_signal()**, **cond_broadcast()**, or when interrupted by delivery of a UNIX signal or a **fork()**.

cond_wait() and **cond_timedwait()** always return with the mutex locked and owned by the calling thread even when returning an error.

A condition signal allows a thread to unblock the next thread waiting on the condition variable, whereas, a condition broadcast allows a thread to unblock all threads waiting on the condition variable.

cond_signal() unblocks one thread that is blocked on the condition variable pointed to by *cvp*.

cond_broadcast() unblocks all threads that are blocked on the condition variable pointed to by *cvp*.

If no threads are blocked on the condition variable, then **cond_signal()** and **cond_broadcast()** have no effect.

Both functions should be called under the protection of the same mutex that is used with the condition variable being signaled. Otherwise, the condition variable may be signaled between the test of the associated condition and blocking in **cond_wait()**. This can cause an infinite wait.

Destroy

The condition destroy functions destroy any state, but not the space, associated with the condition variable.

cond_destroy() destroys any state associated with the condition variable pointed to by *cvp*. The space for storing the condition variable is not freed.

RETURN VALUES

Upon successful completion, these functions return 0. Otherwise, a non-zero value is returned to indicate the error.

ERRORS

These functions may fail if:

EFAULT *cond*, *attr*, *cvp*, *arg*, *abstime*, or *mutex* point to an illegal address.

EINVAL Invalid argument. For **cond_init()**, *type* is not a recognized type. For **cond_timedwait()**, the specified number of seconds, *abstime*, is greater than *current_time* + 100,000,000, where *current_time* is the current time, or the number of nanoseconds is greater than or equal to 1,000,000,000.

The **cond_timedwait()** function may fail if:

ETIME The time specified by *abstime* has passed.

EXAMPLES

EXAMPLE 1 `cond_wait()` is normally used in a loop testing some condition, as follows:

```
(void) mutex_lock(mp);
while (cond == FALSE) {
    (void) cond_wait(cvp, mp);
}
(void) mutex_unlock(mp);
```

EXAMPLE 2 `cond_timedwait()` is also normally used in a loop testing in some conditions. It uses an absolute timeout value as follows:

```
timestruc_t to;
...
(void) mutex_lock(mp);
to.tv_sec = time(NULL) + TIMEOUT;
to.tv_nsec = 0;
while (cond == FALSE) {
    err = cond_timedwait(cvp, mp, &to);
    if (err == ETIMEDOUT) {
        /* timeout, do something */
        break;
    }
}
(void) mutex_unlock(mp);
```

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`fork(2)`, `mmap(2)`, `setitimer(2)`, `shmop(2)`, `condition(3T)`, `mutex(3T)`, `signal(3C)`, `attributes(5)`, `standards(5)`

NOTES

The only policy currently supported is `SCHED_OTHER`. In Solaris, under the `SCHED_OTHER` policy, there is no established order in which threads are unblocked.

If more than one thread is blocked on a condition variable, the order in which threads are unblocked is determined by the scheduling policy. When each thread, unblocked as a result of a `cond_signal()` or `cond_broadcast()`, returns from its call to `cond_wait()` or `cond_timedwait()`, the thread owns the mutex with which it called `cond_wait()` or `cond_timedwait()`. The thread(s) that are unblocked compete for the mutex according to the scheduling policy, and as if each had called `mutex_lock(3T)`.

When `cond_wait()` returns the value of the condition is indeterminate and must be reevaluated.

cond_timedwait() is similar to **cond_wait()** , except that the calling thread will not wait for the condition to become true past the absolute time specified by *abstime* . Note that **cond_timedwait()** may continue to block as it tries to reacquire the mutex pointed to by *mp* , which may be locked by another thread. If *abstime* then **cond_timedwait()** returns because of a timeout, it returns the error code `ETIME` .

| | |
|-----------------------|--|
| NAME | condition – concepts related to condition variables |
| DESCRIPTION | <p>Occasionally, a thread running within a mutex needs to wait for an event, in which case it blocks or sleeps. When a thread is waiting for another thread to communicate its disposition, it uses a condition variable in conjunction with a mutex. Although a mutex is exclusive and the code it protects is sharable (at certain moments), condition variables enable the synchronization of differing events that share a mutex, but not necessarily data. Several condition variables may be used by threads to signal each other when a task is complete, which then allows the next waiting thread to take ownership of the mutex.</p> <p>A condition variable enables threads to atomically block and test the condition under the protection of a mutual exclusion lock (mutex) until the condition is satisfied. If the condition is false, a thread blocks on a condition variable and atomically releases the mutex that is waiting for the condition to change. If another thread changes the condition, it may wake up waiting threads by signaling the associated condition variable. The waiting threads, upon awakening, reacquire the mutex and re-evaluate the condition.</p> |
| Initialize | <p>Condition variables and mutexes should be global. Condition variables that are allocated in writable memory can synchronize threads among processes if they are shared by the cooperating processes (see <code>mmap(2)</code>) and are initialized for this purpose.</p> <p>The scope of a condition variable is either intra-process or inter-process. This is dependent upon whether the argument is passed implicitly or explicitly to the initialization of that condition variable. A condition variable does not need to be explicitly initialized. A condition variable is initialized with all zeros, by default, and its scope is set to within the calling process. For inter-process synchronization, a condition variable must be initialized once, and only once, before use.</p> <p>A condition variable must not be simultaneously initialized by multiple threads or re-initialized while in use by other threads.</p> <p>Condition variables attributes may be set to the default or customized at initialization. POSIX threads even allow the default values to be customized. Establishing these attributes varies depending upon whether POSIX or Solaris threads are used. Similar to the distinctions between POSIX and Solaris thread creation, POSIX condition variables implement the default, intra-process, unless an attribute object is modified for inter-process prior to the initialization of the condition variable. Solaris condition variables also implement as the default, intra-process; however, they set this attribute according to the argument, <i>type</i>, passed to their initialization function.</p> |
| Condition Wait | <p>The condition wait interface allows a thread to wait for a condition and atomically release the associated mutex that it needs to hold to check the</p> |

condition. The thread waits for another thread to make the condition true and that thread's resulting call to `signal` and `wakeup` the waiting thread.

Condition Signaling

A condition signal allows a thread to unblock the next thread waiting on the condition variable, whereas, a condition broadcast allows a thread to unblock all threads waiting on the condition variable.

Destroy

The condition destroy functions destroy any state, but not the space, associated with the condition variable.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`fork(2)`, `mmap(2)`, `setitimer(2)`, `shmop(2)`, `cond_init(3T)`, `cond_wait(3T)`, `cond_timedwait(3T)`, `cond_signal(3T)`, `cond_broadcast(3T)`, `cond_destroy(3T)`, `mutex(3T)`, `pthread_condattr_init(3T)`, `pthread_cond_init(3T)`, `pthread_cond_wait(3T)`, `pthread_cond_timedwait(3T)`, `pthread_cond_signal(3T)`, `pthread_cond_broadcast(3T)`, `pthread_cond_destroy(3T)`, `signal(3C)`, `attributes(5)`, `standards(5)`

NOTES

If more than one thread is blocked on a condition variable, the order in which threads are unblocked is determined by the scheduling policy.

`USYNC_THREAD` does not support multiple mappings to the same logical synch object. If you need to `mmap()` a synch object to different locations within the same address space, then the synch object should be initialized as a shared object `USYNC_PROCESS` for Solaris, and `PTHREAD_PROCESS_PRIVATE` for POSIX.

| | |
|--|--|
| NAME | config_admin, config_change_state, config_private_func, config_test, config_stat, config_list, config_ap_id_cmp, config_unload, config_strerror – configuration administration interface |
| SYNOPSIS | <pre> cc [<i>flag</i>] <i>file</i> -lcfgadm -ldevinfo -ldl [<i>library</i> ...] #include <config_admin.h> cfga_err_t config_change_state(cfga_cmd_t state_change_cmd, int num_ap_ids, char * const * ap_ids, const char * options, struct cfga_confirm * confp, struct cfga_msg * msgp, char ** errstring, cfga_flags_t flags); cfga_err_t config_private_func(const char * function, int num_ap_ids, char * const * ap_ids, const char * options, struct cfga_confirm * confp, struct cfga_msg * msgp, char ** errstring, cfga_flags_t flags); cfga_err_t config_test(int num_ap_ids, char * const * ap_ids, const char * options, struct cfga_msg * msgp, char ** errstring, cfga_flags_t flags); cfga_err_t config_stat(int num_ap_ids, char * const * ap_ids, struct cfga_stat_data * buf, const char * options, char ** errstring); cfga_err_t config_list(struct cfga_stat_data ** ap_id_list, int * nlist, const char * options, char ** errstring); int config_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t ap_id2); void config_unload_libs(); const char * config_strerror(cfga_err_t cfgerinum); </pre> |
| HARDWARE DEPENDENT LIBRARY SYNOPSIS | <p>The config_admin library is a generic interface that is used for dynamic configuration, (DR). Each piece of hardware that supports DR must supply a hardware specific <i>plugin</i> library that contains the entry points listed in this subsection. The generic library will locate and link to the appropriate library to effect DR operations. The interfaces specified in this subsection are really</p> |

"hidden" from users of the generic libraries. It is, however, necessary that writers of the hardware specific plug in libraries know what these interfaces are.

```
cfga_err_t cfga_change_state(cfga_cmd_t state_change_cmd, const char * ap_id, const char * options, struct cfga_confirm * confp, struct cfga_msg * msgp, char ** errstring, cfga_flags_t flags);
```

```
cfga_err_t cfga_private_func(const char * function, const char * ap_id, const char * options, struct cfga_confirm * confp, struct cfga_msg * msgp, char ** errstring, cfga_flags_t flags);
```

```
cfga_err_t cfga_test(const char * ap_id, const char * options, struct cfga_msg * msgp, char ** errstring, cfga_flags_t flags);
```

```
cfga_err_t cfga_stat(const char * ap_id, struct cfga_stat_data * buf, const char * options, char ** errstring);
```

```
cfga_err_t cfga_lib(const char * ap_id, struct cfga_stat_data ** ap_id_list, int * nlist, const char * options, char ** errstring);
```

```
cfga_err_t cfga_help(struct cfga_msg * msgp, const char * options, cfga_flags_t flags);
```

```
int cfga_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t ap_id2);
```

DESCRIPTION

The `config_*` routines provide a hardware independent interface to hardware specific system configuration administration functions. The `cfga_*` routines are provided by hardware specific libraries that are dynamically loaded to handle configuration administration functions in a hardware specific manner.

The `libcfgadm` library is used to provide the services of the `cfgadm(1M)` command. The hardware specific libraries are located in `/usr/platform/${machine}/lib/cfgadm`, `/usr/platform/${arch}/lib/cfgadm`, and `/usr/lib/cfgadm`. The hardware specific library names are derived from the nodename and driver name in device tree nodes that identify attachment points (`DDI_NT_ATTACHMENT_POINT`).

The `config_change_state` routine performs functions that change the state of the system configuration. The `state_change_cmd` can be one of the following: `CFGA_CMD_INSERT`, `CFGA_CMD_REMOVE`, `CFGA_CMD_DISCONNECT`, `CFGA_CMD_CONNECT`, `CFGA_CMD_CONFIGURE` or `CFGA_CMD_UNCONFIGURE`. The `state_change_cmd` `CFGA_CMD_INSERT` is used to prepare for manual insertion or to activate automatic hardware insertion of an occupant. The `state_change_cmd` `CFGA_CMD_REMOVE` is used to prepare for manual removal or activate automatic hardware removal of an occupant. The `state_change_cmd` `CFGA_CMD_DISCONNECT` is used to disable normal communication to or from an occupant in a receptacle. The `state_change_cmd` `CFGA_CMD_CONNECT` is used

to enable communication to or from an occupant in a receptacle. The *state_change_cmd* `CFGA_CMD_CONFIGURE` is used to bring the hardware resources contained on, or attached to, an occupant into the realm of Solaris, allowing use of the occupant's hardware resources by the system. The *state_change_cmd* `CFGA_CMD_UNCONFIGURE` is used to remove the hardware resources contained on, or attached to, an occupant from the realm of Solaris, disallowing further use of the occupant's hardware resources by the system.

The *flags* argument may contain one or both of the defined flags, `CFGA_FLAG_FORCE` and `CFGA_FLAG_VERBOSE`. If the `CFGA_FLAG_FORCE` flag is asserted certain safety checks will be overridden. For example, this may not allow an occupant in the failed condition to be configured, but might allow an occupant in the failing condition to be configured. Acceptance of a force is hardware dependent. If the `CFGA_FLAG_VERBOSE` flag is asserted hardware specific details relating to the operation are output utilizing the `cfga_msg` mechanism.

The `config_private_func` routine invokes private hardware specific functions.

The `config_test` routine is used to initiate testing of the specified attachment point.

The *num_ap_ids* argument specifies the number of *ap_id*s in the *ap_ids* array. The *ap_ids* argument points to an array of *ap_id*s.

The *ap_id* argument points to a single *ap_id*.

The *function* and *options* strings conform to the `getsubopt(3C)` syntax convention and are used to supply hardware specific function or option information. No generic hardware independent functions or options are defined.

The `cfga_confirm` structure referenced by *confp* provides a call-back interface to get permission to proceed should the requested operation require, for example, a noticeable service interruption. The `cfga_confirm` structure includes the following members:

```
int (*confirm)(void *
appdata_ptr
, const char *
message
);
void *appdata_ptr;
```

The `confirm` function is called with two arguments: The generic pointer *appdata_ptr* and the message detailing what requires confirmation. The generic pointer *appdata_ptr* is set to the value passed in in the `cfga_confirm` structure

member `appdata_ptr` and can be used in a graphical user interface to relate the `confirm` function call to the `config_*` call. The `confirm` function should return one (1) to allow the operation to proceed and zero (0) otherwise.

The `cfga_msg` structure referenced by `msgp` provides a call-back interface to output messages from a hardware specific library. In the presence of the `CFG_FLAG_VERBOSE` flag these messages can be informational, otherwise they are restricted to error messages. The `cfga_msg` structure includes the following members:

```
void (*message_routine)(void *
appdata_ptr
, const char *
message
);
void *appdata_ptr;
```

The `message_routine()` function is called with two arguments: The generic pointer `appdata_ptr` and the message. The generic pointer `appdata_ptr` is set to the value passed in in the `cfga_confirm` structure member `appdata_ptr` and can be used in a graphical user interface to relate the `message_routine()` function call to the `config_*` call. The messages must be in the native language specified by the `LC_MESSAGES` locale category; see `setlocale(3C)`.

For some generic errors a hardware specific error message can be returned. The storage for the error message string, including the terminating null character, is allocated by the `config_*` functions using `malloc(3C)` and a pointer to this storage returned through `errstring`. If `errstring` is NULL no error message will be generated or returned. If `errstring` is not NULL and no error message is generated, the pointer referenced by `errstring` will be set to NULL. It is the responsibility of the function calling `config_*` to de-allocate the returned storage using `free(3C)`. The error messages must be in the native language specified by the `LC_MESSAGES` locale category; see `setlocale(3C)`.

The `config_stat` routine provides a way of getting status for an attachment point. The `cfga_stat_data` structure includes the following members:

```
cfga_ap_id_t      ap_log_id;          /* Attachment point logical id */
cfga_ap_id_t      ap_phys_id;         /* Attachment point physical id */
cfga_stat_t       ap_r_state;          /* Receptacle state */
cfga_stat_t       ap_o_state;          /* Occupant state */
cfga_cond_t       ap_cond;             /* Attachment point condition */
cfga_busy_t       ap_busy;             /* Busy indicator */
time_t            ap_status_time;      /* Attachment point last change*/
cfga_info_t       ap_info;             /* Miscellaneous information */
cfga_type_t       ap_type;             /* Occupant type */
```

The types are defined as follows:

```
typedef char  cfga_ap_id_t[CFGGA_AP_ID_LEN];
typedef char  cfga_info_t[CFGGA_INFO_LEN];
typedef char  cfga_type_t[CFGGA_TYPE_LEN];
typedef enum  cfga_cond_t;
typedef enum  cfga_stat_t;
typedef enum  cfga_busy_t;
typedef int   cfga_flags_t;
```

The `ap_log_id` and the `ap_phys_id` fields give the hardware specific logical and physical names of the attachment point. The `ap_busy` field indicates activity is present that may result in changes to state or condition. The `ap_status_time` field gives the time at which either the `ap_r_state`, `ap_o_state` or `ap_cond` fields of the attachment point, last changed. The field `ap_info` is available for the hardware specific code to provide additional information about the attachment point.

The fields `ap_log_id`, `ap_phys_id`, `cfga_info_t` and `cfga_type_t` are null terminated strings. When printing these fields the following format is suggested:

```
printf("%.*s", sizeof(p->ap_log_id), p->ap_log_id);
```

The `config_list` routine provides a way of obtaining the status of all attachment points in the system. The function returns an array of `cfga_stat_data` structures, one for each attachment point in the system. The storage for the array is allocated by the `config_list` function using `malloc(3C)` and a pointer to this storage returned through `ap_id_list`. The number of array elements is returned through `nlist`. It is the responsibility of the function calling `config_list` to de-allocate the returned storage using `free(3C)`.

The `config_ap_id_cmp` function performs a hardware dependent comparison on two `ap_id`s, returning an equal to, less than or greater than indication in the manner of `strcmp(3C)`. Each argument is either a `cfga_ap_id_t` or can be a null terminated string. This function can be used when sorting lists of `ap_id`s, for example with `qsort(3C)`, or when selecting entries from the result of a `config_list` function call.

The `config_unload_libs` function unlinks all previously loaded hardware specific libraries.

The `config_strerror` function can be used to map an error return value to an error message string. See `RETURN VALUES`. The returned string should not be overwritten. `config_strerror` returns `NULL` if `cfgernum` is out-of-range.

RETURN VALUES

The `cfga_help` function can be used request that a hardware specific library output it's localized help message.

The `config_*` and `cfga_*` routines return the following possible values. Additional error information may be returned through *errstring*, if the return code is not `CFGA_OK`. See `DESCRIPTION` for details.

`CFGA_BUSY`

The command was not completed due to an element of the system configuration administration system being busy.

`CFGA_ERROR`

An error occurred during the processing of the requested operation. This error code includes validation of the command arguments by the hardware specific code.

`CFGA_INSUFFICIENT_CONDITION`

Operation failed due to attachment point condition.

`CFGA_INVALID`

The system configuration administration operation requested is not supported on the specified attachment point.

`CFGA_LIB_ERROR`

A procedural error occurred in the library, including failure to obtain process resources such as memory and file descriptors.

`CFGA_NACK`

The command was not completed due to a negative acknowledgement from the *confp* ->`confirm` function.

`CFGA_NO_LIB`

A hardware specific library could not be located using the supplied *ap_id*.

`CFGA_NOTSUPP`

System configuration administration is not supported on the specified attachment point.

CFGA_OK

The command completed as requested.

CFGA_OPNOTSUPP

System configuration administration operation is not supported on this attachment point.

CFGA_PRIV

The caller does not have the required process privileges. For example, if configuration administration is performed through a device driver, the permissions on the device node would be used to control access.

CFGA_SYSTEM_BUSY

The command required a service interruption and was not completed due to a part of the system that could not be quiesced.

ERRORS

Many of the errors returned by the system configuration administration functions are hardware specific. The strings returned in *errstring* may include the following:

attachment point *ap_id* not known

The attachment point detailed in the error message does not exist.

unknown hardware option *option* for *operation*

An unknown option was encountered in the *options* string.

hardware option *option* requires a value

An option in the *options* string should have been of the form *option = value* .

hardware option *option* does not require a value

An option in the *options* string should have been a simple option.

attachment point *ap_id* is not configured

A *config_change_state* command to CFGA_CMD_UNCONFIGURE an occupant was made to an attachment point whose occupant was not in the CFGA_STAT_CONFIGURED state.

attachment point **ap_id** is not unconfigured

A *config_change_state* command requiring an unconfigured occupant was made to an attachment point whose occupant was not in the CFGA_STAT_UNCONFIGURED state.

attachment point **ap_id** condition not satisfactory

A *config_change_state* command was made to an attachment point whose condition prevented the operation.

attachment point **ap_id** in condition **condition** cannot be used

A *config_change_state* operation with force indicated was directed to an attachment point whose condition fails the hardware dependent test.

ATTRIBUTES

See *attributes*(5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|------------------|
| Availability | SUNWcsu, SUNWkvm |
| MT-Level | Safe |

SEE ALSO

cfgadm(1M), *devinfo*(1M), *dlopen*(3X), *dlsym*(3X), *free*(3C), *getsubopt*(3C), *malloc*(3C), *qsort*(3C), *setlocale*(3C), *strcmp*(3C), *libcfdadm*(4), *attributes*(5)

NOTES

Applications using this library should be aware that the underlying implementation may use system services which alter the contents of the external variable *errno* and may use file descriptor resources.

The following code shows the intended error processing when *config_** returns a value other than *CFGA_OK* :

```
void
emit_error(int cfgerrnum, char *estrp)
{
    const char *ep;
    ep = config_strerror(cfgerrnum);
    if (ep == NULL)
```

```
        ep = gettext("configuration administration unknown error");
        if (estrp != NULL && *estrp != '\\0') {
            (void) fprintf(stderr, "%s: %s\
", ep, estrp);
        } else {
            (void) fprintf(stderr, "%s\
", ep);
        }
        if (estrp != NULL)
            free((void *)estrp);
    }
```

Reference should be made to the Hardware Specific Guide for details of System Configuration Administration support.

| | |
|--------------------|---|
| NAME | confstr – get configurable variables |
| SYNOPSIS | <pre>#include <unistd.h> size_t confstr(int name, char *buf, size_t len);</pre> |
| DESCRIPTION | <p>The confstr() function provides a method for applications to get configuration-defined string values. Its use and purpose are similar to the sysconf(3C) function, but it is used where string values rather than numeric values are returned.</p> <p>The <i>name</i> argument represents the system variable to be queried.</p> <p>If <i>len</i> is not 0, and if <i>name</i> has a configuration-defined value, confstr() copies that value into the <i>len</i>-byte buffer pointed to by <i>buf</i>. If the string to be returned is longer than <i>len</i> bytes, including the terminating null, then confstr() truncates the string to <i>len</i>–1 bytes and null-terminates the result. The application can detect that the string was truncated by comparing the value returned by confstr() with <i>len</i>.</p> <p>If <i>len</i> is 0, and <i>buf</i> is a null pointer, then confstr() still returns the integer value as defined below, but does not return the string. If <i>len</i> is 0 but <i>buf</i> is not a null pointer, the result is unspecified.</p> <p>The confstr() function supports the following values for <i>name</i>, defined in <code><unistd.h></code>, for both SPARC and x86:</p> <p><code>_CS_LFS64_CFLAGS</code></p> <p>If <code>_LFS64_LARGEFILE</code> is defined in <code><unistd.h></code>, this value is the set of initial options to be given to the <code>cc</code> and <code>c89</code> utilities to build an application using the Large File Summit transitional compilation environment (see <code>lfcompile64(5)</code>).</p> <p><code>_CS_LFS64_LDFLAGS</code></p> <p>If <code>_LFS64_LARGEFILE</code> is defined in <code><unistd.h></code>, this value is the set of final options to be given to the <code>cc</code> and <code>c89</code> utilities to build an application using the Large File Summit transitional compilation environment (see <code>lfcompile64(5)</code>).</p> <p><code>_CS_LFS64_LIBS</code></p> <p>If <code>_LFS64_LARGEFILE</code> is defined in <code><unistd.h></code>, this value is the set of libraries to be given to the <code>cc</code> and <code>c89</code> utilities to build an application using the Large File Summit transitional compilation environment (see <code>lfcompile64(5)</code>).</p> |

_CS_LFS64_LINTFLAGS

If **_LFS64_LARGEFILE** is defined in `<unistd.h>`, this value is the set of options to be given to the `lint` utility to check application source using the Large File Summit transitional compilation environment (see `1fcompile64(5)`).

_CS_LFS_CFLAGS

If **_LFS_LARGEFILE** is defined in `<unistd.h>`, this value is the set of initial options to be given to the `cc` and `c89` utilities to build an application using the Large File Summit large file compilation environment for 32-bit applications (see `1fcompile(5)`).

_CS_LFS_LDFLAGS

If **_LFS_LARGEFILE** is defined in `<unistd.h>`, this value is the set of final options to be given to the `cc` and `c89` utilities to build an application using the Large File Summit large file compilation environment for 32-bit applications (see `1fcompile(5)`).

_CS_LFS_LIBS

If **_LFS_LARGEFILE** is defined in `<unistd.h>`, this value is the set of libraries to be given to the `cc` and `c89` utilities to build an application using the Large File Summit large file compilation environment for 32-bit applications (see `1fcompile(5)`).

_CS_LFS_LINTFLAGS

If **_LFS_LARGEFILE** is defined in `<unistd.h>`, this value is the set of options to be given to the `lint` utility to check application source using the Large File Summit large file compilation environment for 32-bit applications (see `1fcompile(5)`).

_CS_PATH

If the ISO POSIX.2 standard is supported, this is the value for the `PATH` environment variable that finds all standard utilities. Otherwise the meaning of this value is unspecified.

_CS_XBS5_ILP32_OFF32_CFLAGS

If `sysconf(_SC_XBS5_ILP32_OFF32)` returns `-1` the meaning of this value is unspecified. Otherwise, this value is the set of initial options to be given to the `cc` and `c89` utilities to build an application using a programming model with 32-bit `int`, `long`, `pointer`, and `off_t` types.

`_CS_XBS5_ILP32_OFF32_LDFLAGS`

If `sysconf(_SC_XBS5_ILP32_OFF32)` returns `-1` the meaning of this value is unspecified. Otherwise, this value is the set of final options to be given to the `cc` and `c89` utilities to build an application using a programming model with 32-bit `int`, `long`, `pointer`, and `off_t` types.

`_CS_XBS5_ILP32_OFF32_LIBS`

If `sysconf(_SC_XBS5_ILP32_OFF32)` returns `-1` the meaning of this value is unspecified. Otherwise, this value is the set of libraries to be given to the `cc` and `c89` utilities to build an application using a programming model with 32-bit `int`, `long`, `pointer`, and `off_t` types.

`_CS_XBS5_ILP32_OFF32_LINTFLAGS`

If `sysconf(_SC_XBS5_ILP32_OFF32)` returns `-1` the meaning of this value is unspecified. Otherwise, this value is the set of options to be given to the `lint` utility to check application source using a programming model with 32-bit `int`, `long`, `pointer`, and `off_t` types.

`_CS_XBS5_ILP32_OFFBIG_CFLAGS`

If `sysconf(_SC_XBS5_ILP32_OFFBIG)` returns `-1` the meaning of this value is unspecified. Otherwise, this value is the set of initial options to be given to the `cc` and `c89` utilities to build an application using a programming model with 32-bit `int`, `long`, and `pointer` types, and an `off_t` type using at least 64 bits.

`_CS_XBS5_ILP32_OFFBIG_LDFLAGS`

If `sysconf(_SC_XBS5_ILP32_OFFBIG)` returns `-1` the meaning of this value is unspecified. Otherwise, this value is the set of final options to be given to the `cc` and `c89` utilities to build an application using a programming model with 32-bit `int`, `long`, and `pointer` types, and an `off_t` type using at least 64 bits.

`_CS_XBS5_ILP32_OFFBIG_LIBS`

If `sysconf(_SC_XBS5_ILP32_OFFBIG)` returns `-1` the meaning of this value is unspecified. Otherwise, this value is the set of libraries to be given to the `cc` and `c89` utilities to build an application using a programming model with 32-bit `int`, `long`, and `pointer` types, and an `off_t` type using at least 64 bits.

`_CS_XBS5_ILP32_OFFBIG_LINTFLAGS`

If `sysconf(_SC_XBS5_ILP32_OFFBIG)` returns `-1` the meaning of this value is unspecified. Otherwise, this value is the set of options to be given to the `lint` utility to check an application using a programming model with 32-bit `int`, `long`, and `pointer` types, and an `off_t` type using at least 64 bits.

The `confstr()` function supports the following values for *name*, defined in `<unistd.h>`, for SPARC only:

`_CS_XBS5_LP64_OFF64_CFLAGS`

If `sysconf(_SC_XBS5_LP64_OFF64)` returns `-1` the meaning of this value is unspecified. Otherwise, this value is the set of initial options to be given to the `cc` and `c89` utilities to build an application using a programming model with 64-bit `int`, `long`, `pointer`, and `off_t` types.

`_CS_XBS5_LP64_OFF64_LDFLAGS`

If `sysconf(_SC_XBS5_LP64_OFF64)` returns `-1` the meaning of this value is unspecified. Otherwise, this value is the set of final options to be given to the `cc` and `c89` utilities to build an application using a programming model with 64-bit `int`, `long`, `pointer`, and `off_t` types.

`_CS_XBS5_LP64_OFF64_LIBS`

If `sysconf(_SC_XBS5_LP64_OFF64)` returns `-1` the meaning of this value is unspecified. Otherwise, this value is the set of libraries to be given to the `cc` and `c89` utilities to build an application using a programming model with 64-bit `int`, `long`, `pointer`, and `off_t` types.

`_CS_XBS5_LP64_OFF64_LINTFLAGS`

If `sysconf(_SC_XBS5_LP64_OFF64)` returns `-1` the meaning of this value is unspecified. Otherwise, this value is the set of options to be given to the `lint` utility to check application source using a programming model with 64-bit `int`, `long`, `pointer`, and `off_t` types.

`_CS_XBS5_LPBIG_OFFBIG_CFLAGS`

If `sysconf(_SC_XBS5_LPBIG_OFFBIG)` returns `-1` the meaning of this value is unspecified. Otherwise, this value is the set of initial options to be given to the `cc` and `c89` utilities to build an application using a programming model with an `int` type using at least 32 bits and `long`, `pointer`, and `off_t` types using at least 64 bits.

`_CS_XBS5_LPBIG_OFFBIG_LDFLAGS`

If `sysconf(_SC_XBS5_LPBIG_OFFBIG)` returns `-1` the meaning of this value is unspecified. Otherwise, this value is the set of final options to be given to the `cc` and `c89` utilities to build an application using a programming model with an `int` type using at least 32 bits and `long`, `pointer`, and `off_t` types using at least 64 bits.

`_CS_XBS5_LPBIG_OFFBIG_LIBS`

If `sysconf(_SC_XBS5_LPBIG_OFFBIG)` returns `-1` the meaning of this value is unspecified. Otherwise, this value is the set of libraries to be given to the `cc` and `c89` utilities to build an application using a programming model with an `int` type using at least 32 bits and `long`, `pointer`, and `off_t` types using at least 64 bits.

`_CS_XBS5_LPBIG_OFFBIG_LINTFLAGS`

If `sysconf(_SC_XBS5_LPBIG_OFFBIG)` returns `-1` the meaning of this value is unspecified. Otherwise, this value is the set of options to be given to the `lint` utility to check application source using a programming model with an `int` type using at least 32 bits and `long`, `pointer`, and `off_t` types using at least 64 bits.

RETURN VALUES

If *name* has a configuration-defined value, the `confstr()` function returns the size of buffer that would be needed to hold the entire configuration-defined value. If this return value is greater than *len*, the string returned in *buf* is truncated.

If *name* is invalid, `confstr()` returns 0 and sets `errno` to indicate the error.

If *name* does not have a configuration-defined value, `confstr()` returns 0 and leaves `errno` unchanged.

ERRORS

The `confstr()` function will fail if:

EINVAL The value of the *name* argument is invalid.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Mt-Safe |

SEE ALSO

`pathconf(2)`, `sysconf(3C)`, `attributes(5)`, `lfcompile(5)`,
`lfcompile64(5)`

| | | | | | | | | | | | | | | | |
|----------------------|--|---------------|---|-------------------|--------------------------------|----------------------|---|---------------------|--|-----------------|--|--------------|-------------------------------------|---------------------|--|
| NAME | connect – initiate a connection on a socket | | | | | | | | | | | | | | |
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lsocket -lnsl [<i>library ...</i>] #include <sys/types.h> #include <sys/socket.h></pre> <pre>int connect(int s, const struct sockaddr *name, struct_t namelen);</pre> | | | | | | | | | | | | | | |
| DESCRIPTION | <p>The parameter <i>s</i> is a socket. If it is of type <code>SOCK_DGRAM</code>, connect() specifies the peer with which the socket is to be associated; this address is the address to which datagrams are to be sent if a receiver is not explicitly designated; it is the only address from which datagrams are to be received. If the socket <i>s</i> is of type <code>SOCK_STREAM</code>, connect() attempts to make a connection to another socket. The other socket is specified by <i>name</i>. <i>name</i> is an address in the communication space of the socket. Each communication space interprets the <i>name</i> parameter in its own way. If <i>s</i> is not bound, then it will be bound to an address selected by the underlying transport provider. Generally, stream sockets may successfully connect() only once; datagram sockets may use connect() multiple times to change their association. Datagram sockets may dissolve the association by connecting to a null address.</p> | | | | | | | | | | | | | | |
| RETURN VALUES | <p>If the connection or binding succeeds, 0 is returned. Otherwise, -1 is returned and sets <code>errno</code> to indicate the error.</p> | | | | | | | | | | | | | | |
| ERRORS | <p>The call fails if:</p> <table border="0"> <tr> <td style="padding-right: 20px;">EACCES</td> <td>Search permission is denied for a component of the path prefix of the pathname in <i>name</i>.</td> </tr> <tr> <td>EADDRINUSE</td> <td>The address is already in use.</td> </tr> <tr> <td>EADDRNOTAVAIL</td> <td>The specified address is not available on the remote machine.</td> </tr> <tr> <td>EAFNOSUPPORT</td> <td>Addresses in the specified address family cannot be used with this socket.</td> </tr> <tr> <td>EALREADY</td> <td>The socket is non-blocking and a previous connection attempt has not yet been completed.</td> </tr> <tr> <td>EBADF</td> <td><i>s</i> is not a valid descriptor.</td> </tr> <tr> <td>ECONNREFUSED</td> <td>The attempt to connect was forcefully rejected. The calling program should <code>close(2)</code> the socket descriptor, and issue another <code>socket(3N)</code> call to obtain a new descriptor before attempting another connect() call.</td> </tr> </table> | EACCES | Search permission is denied for a component of the path prefix of the pathname in <i>name</i> . | EADDRINUSE | The address is already in use. | EADDRNOTAVAIL | The specified address is not available on the remote machine. | EAFNOSUPPORT | Addresses in the specified address family cannot be used with this socket. | EALREADY | The socket is non-blocking and a previous connection attempt has not yet been completed. | EBADF | <i>s</i> is not a valid descriptor. | ECONNREFUSED | The attempt to connect was forcefully rejected. The calling program should <code>close(2)</code> the socket descriptor, and issue another <code>socket(3N)</code> call to obtain a new descriptor before attempting another connect() call. |
| EACCES | Search permission is denied for a component of the path prefix of the pathname in <i>name</i> . | | | | | | | | | | | | | | |
| EADDRINUSE | The address is already in use. | | | | | | | | | | | | | | |
| EADDRNOTAVAIL | The specified address is not available on the remote machine. | | | | | | | | | | | | | | |
| EAFNOSUPPORT | Addresses in the specified address family cannot be used with this socket. | | | | | | | | | | | | | | |
| EALREADY | The socket is non-blocking and a previous connection attempt has not yet been completed. | | | | | | | | | | | | | | |
| EBADF | <i>s</i> is not a valid descriptor. | | | | | | | | | | | | | | |
| ECONNREFUSED | The attempt to connect was forcefully rejected. The calling program should <code>close(2)</code> the socket descriptor, and issue another <code>socket(3N)</code> call to obtain a new descriptor before attempting another connect() call. | | | | | | | | | | | | | | |

| | |
|---|---|
| EINPROGRESS | The socket is non-blocking and the connection cannot be completed immediately. It is possible to select (3C) for completion by selecting the socket for writing. However, this is only possible if the socket STREAMS module is the topmost module on the protocol stack with a write service procedure. This will be the normal case. |
| EINTR | The connection attempt was interrupted before any data arrived by the delivery of a signal. |
| EINVAL | <i>namelen</i> is not the size of a valid address for the specified address family. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| EISCONN | The socket is already connected. |
| ELOOP | Too many symbolic links were encountered in translating the pathname in <i>name</i> . |
| ENETUNREACH | The network is not reachable from this host. |
| ENOENT | A component of the path prefix of the pathname in <i>name</i> does not exist. |
| ENOENT | The socket referred to by the pathname in <i>name</i> does not exist. |
| ENOSR | There were insufficient STREAMS resources available to complete the operation. |
| ENXIO | The server exited before the connection was complete. |
| ETIMEDOUT | Connection establishment timed out without establishing a connection. |
| EWouldBLOCK | The socket is marked as non-blocking, and the requested operation would block. |
| The following errors are specific to connecting names in the UNIX domain. These errors may not apply in future versions of the UNIX IPC domain. | |
| ENOTDIR | A component of the path prefix of the pathname in <i>name</i> is not a directory. |
| ENOTSOCK | <i>s</i> is not a socket. |

ENOTSOCK

name is not a socket.

EPROTOTYPE

The file referred to by *name* is a socket of a type other than type *s* (for example, *s* is a SOCK_DGRAM socket, while *name* refers to a SOCK_STREAM socket).

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

close(2), **accept(3N)**, **getsockname(3N)**, **select(3C)**, **socket(3N)**, **attributes(5)**, **socket(5)**

| | |
|--------------------|--|
| NAME | connect – connect a socket |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lxnet [<i>library</i> ...] #include <sys/socket.h></pre> <pre>int connect(int <i>socket</i>, const struct sockaddr *<i>address</i>, socklen_t <i>address_len</i>);</pre> |
| DESCRIPTION | <p>The connect() function requests a connection to be made on a socket. The function takes the following arguments:</p> <p>socket Specifies the file descriptor associated with the socket.</p> <p>address Points to a <code>sockaddr</code> structure containing the peer address. The length and format of the address depend on the address family of the socket.</p> <p>address_len Specifies the length of the <code>sockaddr</code> structure pointed to by the <i>address</i> argument.</p> <p>If the socket has not already been bound to a local address, connect() will bind it to an address which, unless the socket's address family is AF_UNIX, is an unused local address.</p> <p>If the initiating socket is not connection-mode, then connect() sets the socket's peer address, but no connection is made. For SOCK_DGRAM sockets, the peer address identifies where all datagrams are sent on subsequent send(3XN) calls, and limits the remote sender for subsequent recv(3XN) calls. If <i>address</i> is a null address for the protocol, the socket's peer address will be reset.</p> <p>If the initiating socket is connection-mode, then connect() attempts to establish a connection to the address specified by the <i>address</i> argument.</p> <p>If the connection cannot be established immediately and O_NONBLOCK is not set for the file descriptor for the socket, connect() will block for up to an unspecified timeout interval until the connection is established. If the timeout interval expires before the connection is established, connect() will fail and the connection attempt will be aborted. If connect() is interrupted by a signal that is caught while blocked waiting to establish a connection, connect() will fail and set <code>errno</code> to EINTR, but the connection request will not be aborted, and the connection will be established asynchronously.</p> <p>If the connection cannot be established immediately and O_NONBLOCK is set for the file descriptor for the socket, connect() will fail and set <code>errno</code> to EINPROGRESS, but the connection request will not be aborted, and the connection will be established asynchronously. Subsequent calls to connect() for the same socket, before the connection is established, will fail and set <code>errno</code> to EALREADY.</p> |

| | |
|----------------------|---|
| | When the connection has been established asynchronously, <code>select(3C)</code> and <code>poll(2)</code> will indicate that the file descriptor for the socket is ready for writing. |
| | The socket in use may require the process to have appropriate privileges to use the <code>connect()</code> function. |
| USAGE | If <code>connect()</code> fails, the state of the socket is unspecified. Portable applications should close the file descriptor and create a new socket before attempting to reconnect. |
| RETURN VALUES | Upon successful completion, <code>connect()</code> returns 0. Otherwise, <code>-1</code> is returned and <code>errno</code> is set to indicate the error. |
| ERRORS | The <code>connect()</code> function will fail if: |
| | EADDRNOTAVAIL The specified address is not available from the local machine. |
| | EAFNOSUPPORT The specified address is not a valid address for the address family of the specified socket. |
| | EALREADY A connection request is already in progress for the specified socket. |
| | EBADF The <i>socket</i> argument is not a valid file descriptor. |
| | ECONNREFUSED The target address was not listening for connections or refused the connection request. |
| | EFAULT The address parameter can not be accessed. |
| | EINPROGRESS <code>O_NONBLOCK</code> is set for the file descriptor for the socket and the connection cannot be immediately established; the connection will be established asynchronously. |
| | EINTR The attempt to establish a connection was interrupted by delivery of a signal that was caught; the connection will be established asynchronously. |
| | EISCONN The specified socket is connection-mode and is already connected. |
| | ENETUNREACH No route to the network is present. |
| | ENOTSOCK The <i>socket</i> argument does not refer to a socket. |

| | |
|---|--|
| EPROTOTYPE | The specified address has a different type than the socket bound to the specified peer address. |
| ETIMEDOUT | The attempt to connect timed out before a connection was made. |
| If the address family of the socket is AF_UNIX, then connect() will fail if: | |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| ELOOP | Too many symbolic links were encountered in translating the pathname in <i>address</i> . |
| ENAMETOOLONG | A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX characters. |
| ENOENT | A component of the pathname does not name an existing file or the pathname is an empty string. |
| ENOTDIR | A component of the path prefix of the pathname in <i>address</i> is not a directory. |
| The connect() function may fail if: | |
| EACCES | Search permission is denied for a component of the path prefix; or write access to the named socket is denied. |
| EADDRINUSE | Attempt to establish a connection that uses addresses that are already in use. |
| ECONNRESET | Remote host reset the connection request. |
| EHOSTUNREACH | The destination host cannot be reached (probably because the host is down or a remote router cannot reach it). |
| EINVAL | The <i>address_len</i> argument is not a valid length for the address family; or invalid address family in sockaddr structure. |
| ENAMETOOLONG | Pathname resolution of a symbolic link produced an intermediate result whose length exceeds PATH_MAX. |
| ENETDOWN | The local interface used to reach the destination is down. |
| ENOBUFS | No buffer space is available. |

ENOSR There were insufficient STREAMS resources available to complete the operation.

EOPNOTSUPP The socket is listening and can not be connected.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

close(2), **poll(2)**, **accept(3XN)**, **bind(3XN)**, **getsockname(3XN)**, **select(3C)**, **send(3XN)**, **shutdown(3XN)**, **socket(3XN)**, **attributes(5)**

| NAME | ConnectToServer – connect to a DMI service provider | | | | |
|----------------------|---|----------------|-----------------|----------|------|
| SYNOPSIS | <pre>cc [flag ...] file ... -ldmici -ldmimi [library ...] #include <dmi/api.hh> bool_t ConnectToServer(ConnectI *argp, DmiRpcHandle *dmi_rpc_handle);</pre> | | | | |
| DESCRIPTION | <p>The ConnectToServer() function enables a management application or a component instrumentation to connect to a DMI service provider.</p> <p>The <i>argp</i> parameter is an input parameter that uses the following data structure:</p> <pre>struct ConnectIN { char *host; const char *nettype; ServerType servertime; RpcType rpctype; }</pre> <p>The <i>host</i> member indicates the host on which the service provider is running. The default is <i>localhost</i>.</p> <p>The <i>nettype</i> member specifies the type of transport RPC uses. The default is <i>netpath</i>.</p> <p>The <i>servertime</i> member indicates whether the connecting process is a management application or a component instrumentation.</p> <p>The <i>rpctype</i> member specifies the type of RPC, either ONC or DCE. Only ONC is supported in the Solaris 7 release.</p> <p>The <i>dmi_rpc_handle</i> parameter is the output parameter that returns DMI RPC handle.</p> | | | | |
| RETURN VALUES | The ConnectToServer() function returns TRUE if successful, otherwise FALSE. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-level | Safe | | | | |
| SEE ALSO | DisconnectToServer(3X) , attributes(5) | | | | |

| NAME | copylist – copy a file into memory | | | | |
|--------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lgen [<i>library</i> ...] #include <libgen.h> char *copylist(const char *filenm, off_t *szptr);</pre> | | | | |
| DESCRIPTION | <p>The copylist() function copies a list of items from a file into freshly allocated memory, replacing new-lines with null characters. It expects two arguments: a pointer <i>filenm</i> to the name of the file to be copied, and a pointer <i>szptr</i> to a variable where the size of the file will be stored.</p> <p>Upon success, copylist() returns a pointer to the memory allocated. Otherwise it returns NULL if it has trouble finding the file, calling malloc(), or reading the file.</p> | | | | |
| USAGE | The copylist() function has a transitional interface for 64-bit file offsets. See 1f64(5) . | | | | |
| EXAMPLES | <p>EXAMPLE 1 Example of copylist() function.</p> <pre>/* read "file" into buf */ off_t size; char *buf; buf = copylist("file", &size); if (buf) { for (i=0; i<size; i++) if (buf[i]) putchar(buf[i]); else putchar('\n'); } } else { fprintf(stderr, "%s: Copy failed for "file".\n", argv[0]); exit (1); }</pre> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | malloc(3C) , attributes(5) , 1f64(5) | | | | |
| NOTES | When compiling multithreaded applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multithreaded applications. | | | | |

NAME copysign – return magnitude of first argument and sign of second argument

SYNOPSIS cc [*flag ...*] *file ...* -lm [*library ...*]
#include <math.h>

```
double copysign(double x, double y);
```

DESCRIPTION The **copysign()** function returns a value with the magnitude of *x* and the sign of *y*. It produces a NaN with the sign of *y* if *x* is a NaN.

RETURN VALUES The **copysign()** function returns a value with the magnitude of *x* and the sign of *y*.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **attributes(5)**

| | | | | | | | | | | | | | | | | | | | |
|-----------------------|--|----------------------|---|----------------------|---|-----------------------|---|-----------------------|--|-----------------------|---|-----------------------|--|-----------------------|--|-----------------------|---|-----------------------|---|
| NAME | copywin – overlay or overwrite any portion of window | | | | | | | | | | | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int copywin(WINDOW *srcwin, WINDOW *dstwin, int sminrow, int smincol, int dminrow, int dmincol, int dmaxrow, int dmaxcol, int overlay);</pre> | | | | | | | | | | | | | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="vertical-align: top;"><i>srcwin</i></td> <td>Is a pointer to the source window to be copied.</td> </tr> <tr> <td style="vertical-align: top;"><i>dstwin</i></td> <td>Is a pointer to the destination window to be overlaid or overwritten.</td> </tr> <tr> <td style="vertical-align: top;"><i>sminrow</i></td> <td>Is the row coordinate of the upper left corner of the rectangular area on the source window to be copied.</td> </tr> <tr> <td style="vertical-align: top;"><i>smincol</i></td> <td>Is the column coordinate of the upper left corner of the rectangular area on the source window to be copied.</td> </tr> <tr> <td style="vertical-align: top;"><i>dminrow</i></td> <td>Is the row coordinate of the upper left corner of the rectangular area on the destination window to be overlaid or overwritten.</td> </tr> <tr> <td style="vertical-align: top;"><i>dmincol</i></td> <td>Is the column coordinate of the upper left corner of the rectangular area on destination window to be overlaid or overwritten.</td> </tr> <tr> <td style="vertical-align: top;"><i>dmaxrow</i></td> <td>Is the row coordinate of the lower right corner of the rectangular area on the destination window to be overlaid or overwritten.</td> </tr> <tr> <td style="vertical-align: top;"><i>dmaxcol</i></td> <td>Is the column coordinate of the lower right corner of the rectangular area on the destination window to be overlaid or overwritten.</td> </tr> <tr> <td style="vertical-align: top;"><i>overlay</i></td> <td>Is a true or false value that determines whether the destination window is overlaid or overwritten.</td> </tr> </table> | <i>srcwin</i> | Is a pointer to the source window to be copied. | <i>dstwin</i> | Is a pointer to the destination window to be overlaid or overwritten. | <i>sminrow</i> | Is the row coordinate of the upper left corner of the rectangular area on the source window to be copied. | <i>smincol</i> | Is the column coordinate of the upper left corner of the rectangular area on the source window to be copied. | <i>dminrow</i> | Is the row coordinate of the upper left corner of the rectangular area on the destination window to be overlaid or overwritten. | <i>dmincol</i> | Is the column coordinate of the upper left corner of the rectangular area on destination window to be overlaid or overwritten. | <i>dmaxrow</i> | Is the row coordinate of the lower right corner of the rectangular area on the destination window to be overlaid or overwritten. | <i>dmaxcol</i> | Is the column coordinate of the lower right corner of the rectangular area on the destination window to be overlaid or overwritten. | <i>overlay</i> | Is a true or false value that determines whether the destination window is overlaid or overwritten. |
| <i>srcwin</i> | Is a pointer to the source window to be copied. | | | | | | | | | | | | | | | | | | |
| <i>dstwin</i> | Is a pointer to the destination window to be overlaid or overwritten. | | | | | | | | | | | | | | | | | | |
| <i>sminrow</i> | Is the row coordinate of the upper left corner of the rectangular area on the source window to be copied. | | | | | | | | | | | | | | | | | | |
| <i>smincol</i> | Is the column coordinate of the upper left corner of the rectangular area on the source window to be copied. | | | | | | | | | | | | | | | | | | |
| <i>dminrow</i> | Is the row coordinate of the upper left corner of the rectangular area on the destination window to be overlaid or overwritten. | | | | | | | | | | | | | | | | | | |
| <i>dmincol</i> | Is the column coordinate of the upper left corner of the rectangular area on destination window to be overlaid or overwritten. | | | | | | | | | | | | | | | | | | |
| <i>dmaxrow</i> | Is the row coordinate of the lower right corner of the rectangular area on the destination window to be overlaid or overwritten. | | | | | | | | | | | | | | | | | | |
| <i>dmaxcol</i> | Is the column coordinate of the lower right corner of the rectangular area on the destination window to be overlaid or overwritten. | | | | | | | | | | | | | | | | | | |
| <i>overlay</i> | Is a true or false value that determines whether the destination window is overlaid or overwritten. | | | | | | | | | | | | | | | | | | |
| DESCRIPTION | <p>The copywin() function overlays or overwrites windows similar to the overlay(3XC) and overwrite(3XC) functions; however, copywin() allows a finer degree of control on what portion of the window to overlay or overwrite.</p> <p>The parameters <i>smincol</i> and <i>sminrow</i> specify the upper left corner of the rectangular area of the source window to be copied. The <i>dminrow</i> and <i>dmincol</i> parameters specify the upper left corner of the rectangular area of the</p> | | | | | | | | | | | | | | | | | | |

destination window to which the specified portion of the source is to be copied. The *dmaxrow* and *dmaxcol* parameters specify the bottom right corner of the rectangular area of the destination window to which the specified portion of the source is to be copied.

If *overlay* is `TRUE`, only non-blank characters are copied to the destination window; if it is `FALSE`, all characters are copied.

For details on how this function handles overlapping windows with multicolumn characters, see the `Overlapping Windows` section of the `curses(3XC)` man page.

RETURN VALUES

On success, the `copywin()` function returns `OK`. Otherwise, it returns `ERR`.

ERRORS

None.

SEE ALSO

`curses_over(3XC)`, `curses(3XC)`, `newpad(3XC)`, `overlay(3XC)`

NAME | `cos` – cosine function

SYNOPSIS | `cc [flag ...] file ... -lm [library ...]`
`#include <math.h>`
`double cos(double x);`

DESCRIPTION | The `cos()` function computes the cosine of *x*, measured in radians.

RETURN VALUES | Upon successful completion, `cos()` returns the cosine of *x*.
If *x* is NaN or $\pm\text{Inf}$, NaN is returned.

ERRORS | No errors will occur.

ATTRIBUTES | See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | `acos(3M)`, `isnan(3M)`, `sin(3M)`, `tan(3M)`, `attributes(5)`

NAME cosh – hyperbolic cosine function

SYNOPSIS cc [*flag ...*] *file ...* -lm [*library ...*]
#include <math.h>
double **cosh**(double *x*);

DESCRIPTION The **cosh()** function computes the hyperbolic cosine of *x*.

RETURN VALUES Upon successful completion, **cosh()** returns the hyperbolic cosine of *x*.
If the result would cause an overflow, **HUGE_VAL** is returned and **errno** is set to **ERANGE**.
If *x* is NaN, NaN is returned.
For exceptional cases, **matherr(3M)** tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS The **cosh()** function will fail if:
ERANGEThe result would cause an overflow.

USAGE An application wishing to check for error situations should set **errno** to 0 before calling **cosh()**. If **errno** is non-zero on return, or the returned value is NaN, an error has occurred.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **acosh(3M)**, **isnan(3M)**, **matherr(3M)**, **sinh(3M)**, **tanh(3M)**, **attributes(5)**, **standards(5)**

| NAME | crypt – string encoding function | | | | |
|----------------------|--|----------------|-----------------|----------|------|
| SYNOPSIS | <pre>#include <unistd.h> char *crypt(const char *key, const char *salt);</pre> | | | | |
| DESCRIPTION | <p>The crypt() function is a string encoding function, used primarily for password encryption. It is based on a one-way encryption algorithm with variations intended (among other things) to frustrate use of hardware implementations of a key search.</p> <p>The <i>key</i> argument points to a string to be encoded (for example, the user's password.) Only the first eight characters are used; the rest are ignored. The <i>salt</i> is a two-character string chosen from the set [a-zA-Z0-9./]. This string is used to perturb the hashing algorithm in one of 4096 different ways.</p> | | | | |
| RETURN VALUES | <p>Upon successful completion, crypt() returns a pointer to the encoded string. The first two characters of the returned value are those of the <i>salt</i> argument. Otherwise it returns a null pointer and sets <code>errno</code> to indicate the error.</p> <p>In multithreaded applications, the return value is a pointer to thread-specific data.</p> | | | | |
| ERRORS | <p>The crypt() function will fail if:</p> <p>ENOSYS The functionality is not supported on this implementation.</p> | | | | |
| USAGE | <p>The return value of crypt() points to static data that is overwritten by each call.</p> <p>The values returned by this function may not be portable among XSI-conformant systems.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe | | | | |
| SEE ALSO | <p>passwd(1), crypt(3X), encrypt(3C), getpass(3C), setkey(3C), passwd(4), attributes(5)</p> | | | | |

| | |
|--------------------|---|
| NAME | crypt – password and file encryption functions |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lcrypt [<i>library</i> ...] #include <crypt.h> char *crypt(const char *key, const char *salt); void setkey(const char *key); void encrypt(char *block, int flag); char *des_crypt(const char *key, const char *salt); void des_setkey(const char *key); void des_encrypt(char *block, int flag); int run_setkey(int *p, const char *key); int run_crypt(long offset, char *buffer, unsigned int count, int *p); int crypt_close(int *p);</pre> |
| DESCRIPTION | <p>des_crypt() is the password encryption function. It is based on a one-way hashing encryption algorithm with variations intended (among other things) to frustrate use of hardware implementations of a key search.</p> <p><i>key</i> is a user's typed password. <i>salt</i> is a two-character string chosen from the set [a-zA-Z0-9. /]; this string is used to perturb the hashing algorithm in one of 4096 different ways, after which the password is used as the key to encrypt repeatedly a constant string. The returned value points to the encrypted password. The first two characters are the salt itself.</p> <p>The des_setkey() and des_encrypt() entries provide (rather primitive) access to the actual hashing algorithm. The argument of des_setkey() is a character array of length 64 containing only the characters with numerical value 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored, thereby creating a 56-bit key that is set into the machine. This key is the key that will be used with the hashing algorithm to encrypt the string <i>block</i> with the function des_encrypt().</p> <p>The argument to the des_encrypt() entry is a character array of length 64 containing only the characters with numerical value 0 and 1. The argument array is modified in place to a similar array representing the bits of the argument after having been subjected to the hashing algorithm using the key set by des_setkey(). If <i>flag</i> is zero, the argument is encrypted; if non-zero, it is decrypted.</p> <p>Note that decryption is not provided in the international version of crypt(). The international version is part of the C Development Set, and the domestic</p> |

version is part of the Security Administration Utilities. If decryption is attempted with the international version of **des_encrypt()**, an error message is printed.

crypt(), **setkey()**, and **encrypt()** are front-end routines that invoke **des_crypt()**, **des_setkey()**, and **des_encrypt()** respectively.

The routines **run_setkey()** and **run_crypt()** are designed for use by applications that need cryptographic capabilities, such as **ed(1)** and **vi(1)**. **run_setkey()** establishes a two-way pipe connection with the **crypt** utility, using *key* as the password argument. **run_crypt()** takes a block of characters and transforms the cleartext or ciphertext into their ciphertext or cleartext using the **crypt** utility. *offset* is the relative byte position from the beginning of the file that the block of text provided in *block* is coming from. *count* is the number of characters in *block*, and *connection* is an array containing indices to a table of input and output file streams. When encryption is finished, **crypt_close()** is used to terminate the connection with the **crypt** utility.

run_setkey() returns `-1` if a connection with the **crypt** utility cannot be established. This result will occur in international versions of the UNIX system in which the **crypt** utility is not available. If a null key is passed to **run_setkey()**, `0` is returned. Otherwise, `1` is returned. **run_crypt()** returns `-1` if it cannot write output or read input from the pipe attached to **crypt()**. Otherwise it returns `0`.

The program must be linked with the object file access routine library `libcrypt.a`.

RETURN VALUES

In the international version of **crypt()**, a flag argument of `1` to **encrypt()** or **des_encrypt()** is not accepted, and `errno` is set to `ENOSYS` to indicate that the functionality is not available.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

ed(1), **login(1)**, **passwd(1)**, **vi(1)**, **getpass(3C)**, **passwd(4)**, **attributes(5)**

NOTES

The return value in **crypt()** points to static data that are overwritten by each call.

This function is only available with the Solaris Encryption Kit.

| | |
|--------------------|--|
| NAME | cset, csetlen, csetcol, csetno, wcsetno – get information on EUC codesets |
| SYNOPSIS | <pre>#include <euc.h> int csetlen(int codeset); int csetcol(int codeset); int csetno(unsigned char c); #include <wdec.h> int wcsetno(wchar_t pc);</pre> |
| DESCRIPTION | <p>Both csetlen() and csetcol() take a code set number <i>codeset</i>, which must be 0, 1, 2, or 3. The csetlen() function returns the number of bytes needed to represent a character of the given Extended Unix Code (EUC) code set, excluding the single-shift characters SS2 and SS3 for codesets 2 and 3. The csetcol() function returns the number of columns a character in the given EUC code set would take on the display.</p> <p>The csetno() function is implemented as a macro that returns a codeset number (0, 1, 2, or 3) for the EUC character whose first byte is <i>c</i>. For example,</p> <pre>#include<euc.h> ... x+=csetcol(csetno(c));</pre> <p>increments a counter “x” (such as the cursor position) by the width of the character whose first byte is <i>c</i>.</p> <p>The wcsetno() function is implemented as a macro that returns a codeset number (0, 1, 2, or 3) for the given process code character <i>pc</i>. For example,</p> <pre>#include<euc.h> #include<wdec.h> ... x+=csetcol(wcsetno(pc));</pre> <p>increments a counter “x” (such as the cursor position) by the width of the Process Code character <i>pc</i>.</p> |
| USAGE | <p>The cset(), csetlen(), csetcol(), csetno(), and wcsetno() functions can be used safely in multithreaded applications, as long as setlocale(3C) is not being called to change the locale.</p> |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |

SEE ALSO

`setlocale(3C)` `euclen(3C)` , `attributes(5)`

| NAME | ctermid, ctermid_r – generate path name for controlling terminal | | | | |
|--------------------|--|----------------|-----------------|----------|------------------|
| SYNOPSIS | <pre>#include <stdio.h> char * ctermid(char * s); char * ctermid_r(char * s);</pre> | | | | |
| DESCRIPTION | | | | | |
| ctermid() | <p>The ctermid() function generates the path name of the controlling terminal for the current process and stores it in a string.</p> <p>If <i>s</i> is a null pointer, the string is stored in an internal static area whose address is returned and whose contents are overwritten at the next call to ctermid(). Otherwise, <i>s</i> is assumed to point to a character array of at least <code>L_ctermid</code> elements; the path name is placed in this array and the value of <i>s</i> is returned. The constant <code>L_ctermid</code> is defined in the header <code><stdio.h></code>.</p> | | | | |
| ctermid_r() | <p>The ctermid_r() function behaves as ctermid() except that if <i>s</i> is a null pointer, the function returns <code>NULL</code>. This function is as proposed in the POSIX.4a Draft #6 document, and is subject to change to be compliant to the standard when it is accepted.</p> | | | | |
| USAGE | <p>The difference between ctermid() and ttynname(3C) is that ttynname() must be passed a file descriptor and returns the actual name of the terminal associated with that file descriptor, while ctermid() returns a string (<code>/dev/tty</code>) that will refer to the terminal if used as a file name. The ttynname() function is useful only if the process already has at least one file open to a terminal.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>See NOTES below.</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | See NOTES below. |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | See NOTES below. | | | | |
| SEE ALSO | ttynname(3C) , attributes(5) | | | | |
| NOTES | <p>The ctermid() function is unsafe in multithreaded applications. The ctermid_r() function is MT-Safe, and should be used instead.</p> <p>When compiling multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the compile line. This flag should be used only with multithreaded applications.</p> | | | | |

| | |
|--------------------|---|
| NAME | ctime, ctime_r, localtime, localtime_r, gmtime, gmtime_r, asctime, asctime_r, tzset, tzsetwall – convert date and time to string |
| SYNOPSIS | <pre>#include <time.h> char * ctime(const time_t * clock); struct tm * localtime(const time_t * clock); struct tm * gmtime(const time_t * clock); char * asctime(const struct tm * tm); extern time_t timezone, altzone; extern int daylight; extern char *tzname[2]; void tzset(void); void tzsetwall(void); char * ctime_r(const time_t * clock, char * buf, int buflen); struct tm * localtime_r(const time_t * clock, struct tm * res); struct tm * gmtime_r(const time_t * clock, struct tm * res); char * asctime_r(const struct tm * tm, char * buf, int buflen);</pre> |
| POSIX | <pre>cc [flag ...] file ... -D_POSIX_PTHREAD_SEMANTICS [library ...] char * ctime_r(const time_t * clock, char * buf); char * asctime_r(const struct tm * tm, char * buf);</pre> |
| DESCRIPTION | <p>The ctime() function converts the time pointed to by <i>clock</i>, representing the time in seconds since the Epoch (00:00:00 UTC, January 1, 1970), to local time in the form of a 26-character string as shown below. Time zone and daylight savings corrections are made before string generation. The fields are constant width:</p> <pre>Fri Sep 13 00:00:00 1986\ \0</pre> |

The **ctime()** function is equivalent to:

```
asctime(localtime( clock ))
```

The **ctime()**, **asctime()**, **gmtime()**, and **localtime()** functions return values in one of two static objects: a broken-down time structure and an array of `char`. Execution of any of the functions may overwrite the information returned in either of these objects by any of the other functions.

The **ctime_r()** function has the same functionality as **ctime()** except that the caller must supply a buffer *buf* with length *buflen* to store the result; *buf* must be at least 26 bytes. The POSIX **ctime_r()** function does not take a *buflen* parameter.

The **localtime()** and **gmtime()** functions return pointers to `tm` structures (see below). The **localtime()** function corrects for the main time zone and possible alternate (“daylight savings”) time zone; the **gmtime()** function converts directly to Coordinated Universal Time (UTC), which is what the UNIX system uses internally.

The **localtime_r()** and **gmtime_r()** functions have the same functionality as **localtime()** and **gmtime()** respectively, except that the caller must supply a buffer *res* to store the result.

The **asctime()** function converts a `tm` structure to a 26-character string, as shown in the above example, and returns a pointer to the string.

The **asctime_r()** function has the same functionality as **asctime()** except that the caller must supply a buffer *buf* with length *buflen* for the result to be stored. The *buf* argument must be at least 26 bytes. The POSIX **asctime_r()** function does not take a *buflen* parameter. The **asctime_r()** function returns a pointer to *buf* upon success. In case of failure, `NULL` is returned and `errno` is set.

Declarations of all the functions and externals, and the `tm` structure, are in the `<time.h>` header. The members of the `tm` structure are:

```
int    tm_sec;    /* seconds after the minute --- [0, 61] */
\011   /* for leap seconds */
int    tm_min;    /* minutes after the hour --- [0, 59] */
int    tm_hour;   /* hour since midnight --- [0, 23] */
int    tm_mday;   /* day of the month --- [1, 31] */
int    tm_mon;    /* months since January --- [0, 11] */
int    tm_year;   /* years since 1900 */
int    tm_wday;   /* days since Sunday --- [0, 6] */
int    tm_yday;   /* days since January 1 --- [0, 365] */
```

```
int    tm_isdst; /* flag for alternate daylight savings time */
```

The value of `tm_isdst` is positive if daylight savings time is in effect, zero if daylight savings time is not in effect, and negative if the information is not available. Previously, the value of `tm_isdst` was defined as non-zero if daylight savings was in effect.

The external `time_t` variable `altzone` contains the difference, in seconds, between Coordinated Universal Time and the alternate time zone. The external variable `timezone` contains the difference, in seconds, between UTC and local standard time. The external variable `daylight` indicates whether time should reflect daylight savings time. Both `timezone` and `altzone` default to 0 (UTC). The external variable `daylight` is non-zero if an alternate time zone exists. The time zone names are contained in the external variable `tzname`, which by default is set to:

```
char *tzname[2] = { "GMT", "" };
```

These functions know about the peculiarities of this conversion for various time periods for the U.S. (specifically, the years 1974, 1975, and 1987). They will handle the new daylight savings time starting with the first Sunday in April, 1987.

The `tzset()` function uses the contents of the environment variable `TZ` to override the value of the different external variables. It is called by `asctime()` and may also be called by the user. See `environ(5)` for a description of the `TZ` environment variable.

Starting and ending times are relative to the current local time zone. If the alternate time zone start and end dates and the time are not provided, the days for the United States that year will be used and the time will be 2 AM. If the start and end dates are provided but the time is not provided, the time will be 2 AM. The effects of `tzset()` change the values of the external variables `timezone`, `altzone`, `daylight`, and `tzname`.

Note that in most installations, `TZ` is set to the correct value by default when the user logs on, using the local `/etc/default/init` file (see `TIMEZONE(4)`).

The `tzsetwall()` function sets things up so that `localtime()` returns the best available approximation of local wall clock time.

ERRORS

The `ctime_r()` and `asctime_r()` functions will fail if the following is true:

ERANGE The length of the buffer supplied by the caller is not large enough to store the result.

USAGE

These functions are included for compatibility with older implementations, and do not support localized date and time formats.

EXAMPLES

EXAMPLE 1 Examples of the `tzset()` function.

The `tzset()` function scans the contents of the environment variable and assigns the different fields to the respective variable. For example, the most complete setting for New Jersey in 1986 could be

```
EST5EDT4,116/2:00:00,298/2:00:00
```

or simply

```
EST5EDT
```

An example of a southern hemisphere setting such as the Cook Islands could be

```
KDT9:30KST10:00,63/5:00,302/20:00
```

In the longer version of the New Jersey example of TZ, `tzname [0]` is EST, `timezone` will be set to 5*60*60, `tzname [1]` is EDT, `altzone` will be set to 4*60*60, the starting date of the alternate time zone is the 117th day at 2 AM, the ending date of the alternate time zone is the 299th day at 2 AM (using zero-based Julian days), and `daylight` will be set positive. Starting and ending times are relative to the current local time zone. If the alternate time zone start and end dates and the time are not provided, the days for the United States that year will be used and the time will be 2 AM. If the start and end dates are provided but the time is not provided, the time will be 2 AM. The effects of `tzset()` are thus to change the values of the external variables `timezone`, `altzone`, `daylight`, and `tzname`. The `ctime()`, `localtime()`, `mktime()`, and `strftime()` functions will also update these external variables as if they had called `tzset()` at the time specified by the `time_t` or `struct tm` value that they are converting.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |
| CSI | Enabled |

SEE ALSO

`time(2)`, `Intro(3)`, `getenv(3C)`, `mktime(3C)`, `printf(3S)`, `putenv(3C)`, `setlocale(3C)`, `strftime(3C)`, `TIMEZONE(4)`, `attributes(5)`, `environ(5)`

NOTES

When compiling multithreaded programs, see **Intro(3)** , *Notes On Multithreaded Applications* .

The return values for **ctime()** , **localtime()** , and **gmtime()** point to static data whose content is overwritten by each call.

Setting the time during the interval of change from `timezone` to `altzone` or vice versa can produce unpredictable results. The system administrator must change the Julian start and end days annually.

The **asctime()** , **ctime()** , **gmtime()** , and **localtime()** functions are unsafe in multithread applications. The **asctime_r()** and **gmtime_r()** functions are MT-Safe. The **ctime_r()** , **localtime_r()** , **tzset()** , and **tzsetwall()** functions are MT-Safe in multithread applications, as long as no user-defined function directly modifies one of the following variables: `timezone` , `altzone` , `daylight` , and `tzname` . These four variables are not MT-Safe to access. They are modified by the **tzset()** function in an MT-Safe manner. The **mktime()** , **localtime_r()** , and **ctime_r()** functions call **tzset()** .

Solaris 2.4 and earlier releases provided definitions of the **ctime_r()** , **localtime_r()** , **gmtime_r()** , and **asctime_r()** functions as specified in POSIX.1c Draft 6. The final POSIX.1c standard changed the interface for **ctime_r()** and **asctime_r()** . Support for the Draft 6 interface is provided for compatibility only and may not be supported in future releases. New applications and libraries should use the POSIX standard interface.

For POSIX.1c-compliant applications, the `_POSIX_PTHREAD_SEMANTICS` and `_REENTRANT` flags are automatically turned on by defining the `_POSIX_C_SOURCE` flag with a value $\geq 199506L$.

| | |
|--------------------|---|
| NAME | ctype, isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii – character handling |
| SYNOPSIS | <pre>#include <ctype.h> int isalpha(int c); int isupper(int c); int islower(int c); int isdigit(int c); int isxdigit(int c); int isalnum(int c); int isspace(int c); int ispunct(int c); int isprint(int c); int isgraph(int c); int iscntrl(int c); int isascii(int c);</pre> |
| DESCRIPTION | <p>These macros classify character-coded integer values. Each is a predicate returning non-zero for true, 0 for false. The behavior of these macros, except isascii(), is affected by the current locale (see setlocale(3C)). To modify the behavior, change the <code>LC_TYPE</code> category in setlocale(), that is, <code>setlocale(LC_CTYPE, newlocale)</code>. In the "C" locale, or in a locale where character type information is not defined, characters are classified according to the rules of the US-ASCII 7-bit coded character set.</p> <p>The macro isascii() is defined on all integer values; the rest are defined only where the argument is an <code>int</code>, the value of which is representable as an <code>unsigned char</code>, or <code>EOF</code>, which is defined by the <code><stdio.h></code> header and represents end-of-file.</p> <p>Functions exist for all the macros defined below. To get the function form, the macro name must be undefined (for example, <code>#undef isdigit</code>).</p> <p>For macros described with <code>Default</code> and <code>Standard-conforming</code> versions, standard-conforming behavior will be provided for standard-conforming applications (see standards(5)) and for applications that define <code>__XPG4_CHAR_CLASS__</code> before including <code><ctype.h></code>.</p> |

| | | |
|---------------------|-------------------|---|
| Default | isalpha() | Tests for any character for which isupper() or islower() is true. |
| Standard-conforming | isalpha() | Tests for any character for which isupper() or islower() is true, or any character that is one of the current locale-defined set of characters for which none of iscntrl() , isdigit() , ispunct() , or isspace() is true. In "C" locale, isalpha() returns true only for the characters for which isupper() or islower() is true. |
| | isupper() | Tests for any character that is an upper-case letter or is one of the current locale-defined set of characters for which none of iscntrl() , isdigit() , ispunct() , isspace() , or islower() is true. In the "C" locale, isupper() returns true only for the characters defined as upper-case ASCII characters. |
| | islower() | Tests for any character that is a lower-case letter or is one of the current locale-defined set of characters for which none of iscntrl() , isdigit() , ispunct() , isspace() , or isupper() is true. In the "C" locale, islower() returns true only for the characters defined as lower-case ASCII characters. |
| | isdigit() | Tests for any decimal-digit character. |
| | isxdigit() | Tests for any hexadecimal-digit character ([0-9] , [A-F] , or [a-f]). |
| Default | isxdigit() | Tests for any hexadecimal-digit character ([0-9] , [A-F] , or [a-f]). |
| Standard-conforming | isxdigit() | Tests for any hexadecimal-digit character ([0-9] , [A-F] , or [a-f]) or the current locale-defined sets of characters representing the hexadecimal digits 10 to 15 inclusive). In the "C" locale, only 0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f are included. |
| | isalnum() | Tests for any character for which isalpha() or isdigit() is true (letter or digit). |

| | | |
|----------------------|------------------|---|
| | isspace() | Tests for any space, tab, carriage-return, newline, vertical-tab or form-feed (standard white-space characters) or for one of the current locale-defined set of characters for which isalnum() is false. In the C locale, isspace() returns true only for the standard white-space characters. |
| | ispunct() | Tests for any printing character which is neither a space (" ") nor a character for which isalnum() or iscntrl() is true. |
| Default | isprint() | Tests for any character for which ispunct() , isupper() , islower() , isdigit() , and the space character (" ") is true. |
| Standard-conforming | isprint() | Tests for any character for which iscntrl() is false, and isalnum() , isgraph() , ispunct() , the space character (" ") , and the characters in the current locale-defined "print" class are true. |
| Default | isgraph() | Tests for any character for which ispunct() , isupper() , islower() , and isdigit() is true. |
| Standard-conforming | isgraph() | Tests for any character for which isalnum() and ispunct() are true, or any character in the current locale-defined "graph" class which is neither a space (" ") nor a character for which iscntrl() is true. |
| | iscntrl() | Tests for any "control character" as defined by the character set. |
| | isascii() | Tests for any ASCII character, code between 0 and 0177 inclusive. |
| RETURN VALUES | | If the argument to any of the character handling macros is not in the domain of the function, the result is undefined. Otherwise, the macro/function will return non-zero if the classification is <code>TRUE</code> , and 0 for <code>FALSE</code> . |
| USAGE | | The isdigit() , isxdigit() , islower() , isupper() , isalpha() , isalnum() , isspace() , iscntrl() , ispunct() , isprint() , isgraph() , and isascii() macros can be used safely in multithreaded applications, as long as setlocale(3C) is not being called to change the locale. |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |
| CSI | Enabled |

SEE ALSO

setlocale(3C), **stdio(3S)**, **ascii(5)**, **environ(5)**, **standards(5)**

| | |
|--------------------|--|
| NAME | curs_addch, addch, waddch, mvaddch, mvwaddch, echochar, wechochar – add a character (with attributes) to a curses window and advance cursor |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ..] #include <curses.h> int addch(chtype <i>ch</i>); int waddch(WINDOW * <i>win</i>, chtype <i>ch</i>); int mvaddch(int <i>y</i>, int <i>x</i>, chtype <i>ch</i>); int mvwaddch(WINDOW * <i>win</i>, int <i>y</i>, int <i>x</i>, chtype <i>ch</i>); int echochar(chtype <i>ch</i>); int wechochar(WINDOW * <i>win</i>, chtype <i>ch</i>);</pre> |
| DESCRIPTION | <p>With the addch() , waddch() , mvaddch() , and mvwaddch() routines, the character <i>ch</i> is put into the window at the current cursor position of the window and the position of the window cursor is advanced. Its function is similar to that of putchar() . At the right margin, an automatic newline is performed. At the bottom of the scrolling region, if scrollok() is enabled, the scrolling region is scrolled up one line.</p> <p>If <i>ch</i> is a tab, newline, or backspace, the cursor is moved appropriately within the window. A newline also does a clrtoeol() before moving. Tabs are considered to be at every eighth column. If <i>ch</i> is another control character, it is drawn in the ^ <i>X</i> notation. Calling winch() after adding a control character does not return the control character, but instead returns the representation of the control character. See curs_inch(3X) .</p> <p>Video attributes can be combined with a character by OR-ing them into the parameter. This results in these attributes also being set. (The intent here is that text, including attributes, can be copied from one place to another using inch() and addch() .) (see standout() , predefined video attribute constants, on the curs_attr(3X) page).</p> <p>The echochar() and wechochar() routines are functionally equivalent to a call to addch() followed by a call to refresh() , or a call to waddch followed by a</p> |

call to **wrefresh()** . The knowledge that only a single character is being output is taken into consideration and, for non-control characters, a considerable performance gain might be seen by using these routines instead of their equivalents.

Line Graphics

The following variables may be used to add line drawing characters to the screen with routines of the **addch()** family. When variables are defined for the terminal, the `A_ALTCHARSET` bit is turned on (see `curs_attr(3X)`). Otherwise, the default character listed below is stored in the variable. The names chosen are consistent with the VT100 nomenclature.

| Name | Default | Glyph Description |
|--------------|---------|-------------------------|
| ACS_ULCORNER | + | upper left-hand corner |
| ACS_LLCORNER | + | lower left-hand corner |
| ACS_URCORNER | + | upper right-hand corner |
| ACS_LRCORNER | + | lower right-hand corner |
| ACS_RTEE | + | right tee (-) |
| ACS_LTEE | + | left tee ((br- |
| ACS_BTEE | + | bottom tee () |
| ACS_TTEE | + | top tee () |
| ACS_HLINE | - | horizontal line |
| ACS_VLINE | | vertical line |
| ACS_PLUS | + | plus |
| ACS_S1 | - | scan line 1 |
| ACS_S9 | - | scan line 9 |
| ACS_DIAMOND | + | diamond |
| ACS_CKBOARD | : | checker board (stipple) |
| ACS_DEGREE | ' | degree symbol |
| ACS_PLMINUS | # | plus/minus |
| ACS_BULLET | o | bullet |
| ACS_LARROW | < | arrow pointing left |
| ACS_RARROW | > | arrow pointing right |
| ACS_DARROW | v | arrow pointing down |
| ACS_UARROW | ^ | arrow pointing up |
| ACS_BOARD | # | board of squares |

| Name | Default | Glyph Description |
|---------------|---------|--------------------|
| ACS_LANTERN | # | lantern symbol |
| ACS_BLOCK\011 | # | solid square block |

RETURN VALUES

All routines return the integer `ERR` upon failure and an integer value other than `ERR` upon successful completion.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`curs_attr(3X)`, `curs_clear(3X)`, `curs_inch(3X)`, `curs_outopts(3X)`, `curs_refresh(3X)`, `curses(3X)`, `putc(3S)`, `attributes(5)`

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

Note that `addch()`, `mvaddch()`, `mvwaddch()`, and `echochar()` may be macros.

| | |
|----------------------|--|
| NAME | curs_addchstr, addchstr, addchnstr, waddchstr, waddchnstr, mvaddchstr, mvaddchnstr, mvwaddchstr, mvwaddchnstr – add string of characters and attributes to a curses window |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> int addchstr(chtype * <i>chstr</i>); int addchnstr(chtype * <i>chstr</i>, int <i>n</i>); int waddchstr(WINDOW * <i>win</i>, chtype * <i>chstr</i>); int waddchnstr(WINDOW * <i>win</i>, chtype * <i>chstr</i>, int <i>n</i>); int mvaddchstr(int <i>y</i>, int <i>x</i>, chtype * <i>chstr</i>); int mvaddchnstr(int <i>y</i>, int <i>x</i>, chtype * <i>chstr</i>, int <i>n</i>); int mvwaddchstr(WINDOW * <i>win</i>, int <i>y</i>, int <i>x</i>, chtype * <i>chstr</i>); int mvwaddchnstr(WINDOW * <i>win</i>, int <i>y</i>, int <i>x</i>, chtype * <i>chstr</i>, int <i>n</i>); </pre> |
| DESCRIPTION | <p>All of these routines copy <i>chstr</i> directly into the window image structure starting at the current cursor position. The four routines with <i>n</i> as the last argument copy at most <i>n</i> elements, but no more than will fit on the line. If <i>n</i> = -1 then the whole string is copied, to the maximum number that fit on the line.</p> <p>The position of the window cursor is not advanced. These routines works faster than waddnstr() (see curs_addstr(3X)) because they merely copy <i>chstr</i> into the window image structure. On the other hand, care must be taken when using these functions because they do not perform any kind of checking (such as for the newline character), they do not advance the current cursor position, and they truncate the string, rather than wrapping it around to the next line.</p> |
| RETURN VALUES | All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion. |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `curl_addstr(3X)`, `curses(3X)`, `attributes(5)`

NOTES The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

Note that all routines except `waddchnstr()` and `waddchstr()` may be macros.

| NAME | curs_addstr, addstr, addnstr, waddstr, waddnstr, mvaddstr, mvaddnstr, mvwaddstr, mvwaddnstr – add a string of characters to a curses window and advance cursor | | | | |
|----------------------|---|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> int addstr(char * <i>str</i>); int addnstr(char * <i>str</i>, int <i>n</i>); int waddstr(WINDOW * <i>win</i>, char * <i>str</i>); int waddnstr(WINDOW * <i>win</i>, char * <i>str</i>, int <i>n</i>); int mvaddstr(int <i>y</i>, int <i>x</i>, char * <i>str</i>); int mvaddnstr(int <i>y</i>, int <i>x</i>, char * <i>str</i>, int <i>n</i>); int mvwaddstr(WINDOW * <i>win</i>, int <i>y</i>, int <i>x</i>, char * <i>str</i>); int mvwaddnstr(WINDOW * <i>win</i>, int <i>y</i>, int <i>x</i>, char * <i>str</i>, int <i>n</i>); </pre> | | | | |
| DESCRIPTION | All of these routines write all the characters of the null terminated character string <i>str</i> on the given window. It is similar to calling waddch() once for each character in the string. The four routines with <i>n</i> as the last argument write at most <i>n</i> characters. If <i>n</i> is negative, then the entire string will be added. | | | | |
| RETURN VALUES | All routines return the integer <code>ERR</code> upon failure and an integer value other than <code>ERR</code> upon successful completion. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | curs_addch(3X) , curses(3X) , attributes(5) | | | | |

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

Note that all routines except **waddstr()** and **waddnstr()** may not be macros.

| | |
|--------------------|--|
| NAME | curs_addwch, addwch, waddwch, mvaddwch, mvwaddwch, echowchar, wechowchar – add a wchar_t character (with attributes) to a curses window and advance cursor |
| SYNOPSIS | <pre>cc [flag...] file... -lcurses [library...] #include<curses.h> int addwch(chtype wch); int waddwch(WINDOW * win, chtype wch); int mvaddwch(int y, int x, chtype wch); int mvwaddwch(WINDOW * win, int y, int x, chtype wch); int echowchar(chtype wch); int wechowchar(WINDOW * win, chtype wch);</pre> |
| DESCRIPTION | <p>The addwch() , waddwch() , mvaddwch() , and mvwaddwch() routines put the character <i>wch</i> , holding a <code>wchar_t</code> character, into the window at the current cursor position of the window and advance the position of the window cursor. Their function is similar to that of putwchar(3S) in the C multibyte library. At the right margin, an automatic newline is performed. At the bottom of the scrolling region, if <code>scrollok</code> is enabled, the scrolling region is scrolled up one line.</p> <p>If <i>wch</i> is a tab, newline, or backspace, the cursor is moved appropriately within the window. A newline also does a clrtoeol(3X) before moving. Tabs are considered to be at every eighth column. If <i>wch</i> is another control character, it is drawn in the <code>^ X</code> notation. Calling winwch(3X) after adding a control character does not return the control character, but instead returns the representation of the control character.</p> <p>Video attributes can be combined with a <code>wchar_t</code> character by OR-ing them into the parameter. This results in these attributes also being set. (The intent here is that text, including attributes, can be copied from one place to another using inwch() and addwch() .) See standout(3X) , predefined video attribute constants.</p> <p>The echowchar() and wechowchar() routines are functionally equivalent to a call to addwch() followed by a call to refresh(3X) , or a call to waddwch() followed by a call to wrefresh(3X) . The knowledge that only a single character is being output is taken into consideration and, for non-control characters, a considerable performance gain might be seen by using these routines instead of their equivalents.</p> |

Line Graphics

The following variables may be used to add line drawing characters to the screen with routines of the **addwch()** family. When variables are defined for the terminal, the **A_ALTCHARSET** bit is turned on. (See **curs_attr(3X)**). Otherwise, the default character listed below is stored in the variable. The names chosen are consistent with the VT100 nomenclature.

| Name | Default | Glyph Description |
|--------------|---------|-------------------------|
| ACS_ULCORNER | + | upper left-hand corner |
| ACS_LLCORNER | + | lower left-hand corner |
| ACS_URCORNER | + | upper right-hand corner |
| ACS_LRCORNER | + | lower right-hand corner |
| ACS_RTEE | + | right tee |
| ACS_LTEE | + | left tee |
| ACS_BTEE | + | bottom tee |
| ACS_TTEE | + | top tee |
| ACS_HLINE | - | horizontal line |
| ACS_VLINE | | vertical line |
| ACS_PLUS | + | plus |
| ACS_S1 | - | scan line 1 |
| ACS_S9 | - | scan line 9 |
| ACS_DIAMOND | + | diamond |
| ACS_CKBOARD | : | checker board (stipple) |
| ACS_DEGREE | ' | degree symbol |
| ACS_PLMINUS | # | plus/minus |
| ACS_BULLET | o | bullet |
| ACS_LARROW | < | arrow pointing left |
| ACS_RARROW | > | arrow pointing right |
| ACS_DARROW | v | arrow pointing down |
| ACS_UARROW | ^ | arrow pointing up |
| ACS_BOARD | # | board of squares |
| ACS_LANTERN | # | lantern symbol |
| ACS_BLOCK | # | solid square block |

RETURN VALUE All routines return the integer `ERR` upon failure and an integer value other than `ERR` upon successful completion, unless otherwise noted in the preceding routine descriptions.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `putwchar(3S)`, `clrtoeol(3X)`, `curses(3X)`, `curs_attr(3X)`, `curs_inwch(3X)`, `curs_outopts(3X)`, `refresh(3X)`, `standout(3X)`, `winwch(3X)`, `wrefresh(3X)`, `attributes(5)`

NOTES The header file `<curses.h>` automatically includes the header files `<stdio.h>`, `<unctrl.h>` and `<widec.h>`.

Note that `addwch()`, `mvaddwch()`, `mvwaddwch()`, and `echowchar()` may be macros.

None of these routines can use the color attribute in `chtype`.

NAME curs_addwchstr, addwchstr, addwchnstr, waddwchstr, waddwchnstr, mvaddwchstr, mvaddwchnstr, mvwaddwchstr, mvwaddwchnstr – add string of wchar_t characters (and attributes) to a curses window

SYNOPSIS cc [*flag...*] *file...* -lcurses [*library...*]

```
#include<curses.h>

int addwchstr(chtype * wchstr);

int addwchnstr(chtype * wchstr, int n);

int waddwchstr(WINDOW * win, chtype * wchstr);

int waddwchnstr(WINDOW * win, chtype * wchstr, int n);

int mvaddwchstr(int y, int x, chtype * wchstr);

int mvaddwchnstr(int y, int x, chtype * wchstr, int n);

int mvwaddwchstr(WINDOW * win, int y, int x, chtype * wchstr);

int mvwaddwchnstr(WINDOW * win, int y, int x, chtype * wchstr, int n);
```

DESCRIPTION All of these routines copy *wchstr*, which points to a string of wchar_t characters, directly into the window image structure starting at the current cursor position. The four routines with *n* as the last argument copy at most *n* elements, but no more than will fit on the line. If *n* = -1 then the whole string is copied, to the maximum number that fit on the line.

The position of the window cursor is not advanced. These routines work faster than **waddnwstr(3X)** because they merely copy *wchstr* into the window image structure. On the other hand, care must be taken when using these functions because they don't perform any kind of checking (such as for the newline character), they do not advance the current cursor position, and they truncate the string, rather than wrapping it around to the new line.

RETURN VALUE All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion, unless otherwise noted in the preceding routine descriptions.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO **curses(3X)**, **waddnwstr(3X)**, **attributes(5)**

NOTES | The header file `<curses.h>` automatically includes the header files `<stdio.h>` , `<unctrl.h>` and `<widec.h>` .

Note that all routines except `waddwchnstr()` may be macros.

None of these routines can use the color attribute in `chtype` .

NAME curs_addwstr, addwstr, addnwstr, waddwstr, waddnwstr, mvaddwstr, mvaddnwstr, mvwaddwstr, mvwaddnwstr – add a string of wchar_t characters to a curses window and advance cursor

SYNOPSIS cc [flag...] file... -lcurses [library...]

```
#include<curses.h>

int addwstr(wchar_t * wstr);

int addnwstr(wchar_t * wstr, int n);

INT WADDWSTR(WINDOW * WIN, wchar_t * wstr);

int waddnwstr(WINDOW * win, wchar_t * wstr, int n);

int mvaddwstr(int y, int x, wchar_t * wstr);

int mvaddnwstr(int y, int x, wchar_t * wstr, int n);

int mvwaddwstr(WINDOW * win, int y, int x, wchar_t * wstr);

int mvwaddnwstr(WINDOW * win, int y, int x, wchar_t * wstr, int n);
```

DESCRIPTION All of these routines write all the characters of the null-terminated wchar_t character string wstr on the given window. The effect is similar to calling waddwch(3X) once for each wchar_t character in the string. The four routines with n as the last argument write at most n wchar_t characters. If n is negative, then the entire string will be added.

RETURN VALUE All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO curses(3X) , waddwch(3X) , attributes(5)

NOTES The header file <curses.h> automatically includes the header files <stdio.h> , <nctrl.h> and <widec.h> .

Note that all of these routines except waddwstr() and waddnwstr() may be macros.

| | |
|---------------------|--|
| NAME | curs_alecompat, movenextch, wmovenextch, moveprevch, wmoveprevch, adjcurspos, wadjcurspos – these functions are added to ALE curses library for moving the cursor by character. |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ..] #include <curses.h> int movenextch(void); int wmovenextch(WINDOW * win); int moveprevch(void); int wmoveprevch(WINDOW * win); int adjcurspos(void); int wadjcurspos(WINDOW * win);</pre> |
| DESCRIPTION | <p>movenextch() and wmovenextch() move the cursor to the next character to the right. If the next character is a multicolumn character, the cursor is positioned on the first (left-most) column of that character. The new cursor position will be on the next character, even if the cursor was originally positioned on the left-most column of a multicolumn character. Note that the simple cursor increment (<code>++x</code>) does not guarantee movement to the next character, if the cursor was originally positioned on a multicolumn character. getyx(3X) can be used to find the new position.</p> <p>moveprevch() and wmoveprevch() routines are the opposite of movenextch() and wmovenextch() , moving the cursor to the left-most column of the previous character.</p> <p>adjcurspos() and wadjcurspos() move the cursor to the first(left-most) column of the multicolumn character that the cursor is presently on. If the cursor is already on the first column, or if the cursor is on a single-column character, these routines will have no effect.</p> |
| RETURN VALUE | All routines return the integer <code>ERR</code> upon failure and an integer value other than <code>ERR</code> upon successful completion. |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

curses(3X), **getyx(3X)**, **attributes(5)**

NOTES

The header file `<curses.h>` automatically includes the header files `<stdio.h>`, `<unctrl.h>` and `<wdec.h>`.

Note that `movenextch()`, `moveprevch()`, and `adjcurspos()` may be macros.

| | |
|--------------------|--|
| NAME | curs_attr, attroff, wattroff, attron, wattron, attrset, wattrset, standend, wstandend, standout, wstandout – curses character and window attribute control routines |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> int attroff(int <i>attrs</i>); int wattroff(WINDOW * <i>win</i>, int <i>attrs</i>); int attron(int <i>attrs</i>); int wattron(WINDOW * <i>win</i>, int <i>attrs</i>); int attrset(int <i>attrs</i>); int wattrset(WINDOW * <i>win</i>, int <i>attrs</i>); int standend(void); int wstandend(WINDOW * <i>win</i>); int standout(void); int wstandout(WINDOW * <i>win</i>);</pre> |
| DESCRIPTION | <p>All of these routines manipulate the current attributes of the named window. The current attributes of a window are applied to all characters that are written into the window with waddch(), waddstr(), and wprintw(). Attributes are a property of the character, and move with the character through any scrolling and insert/delete line/character operations. To the extent possible on the particular terminal, they are displayed as the graphic rendition of characters put on the screen.</p> <p>The routine attrset() sets the current attributes of the given window to <i>attrs</i>. The routine attroff() turns off the named attributes without turning any other attributes on or off. The routine attron() turns on the named attributes without affecting any others. The routine standout() is the same as <code>attron(A_STANDOUT)</code>. The routine standend() is the same as <code>attrset()</code>, that is, it turns off all attributes.</p> |

Attributes | The following video attributes, defined in `<curses.h>`, can be passed to the routines `attron()`, `attroff()`, and `attrset()`, or OR-ed with the characters passed to `addch()`.

A_STANDOUT | Best highlighting mode of the terminal

A_UNDERLINE | Underlining

A_REVERSE | Reverse video

A_BLINK | Blinking

A_DIM | Half bright

A_BOLD | Extra bright or bold

A_ALTCHARSET | Alternate character set

A_CHARTEXT | \011Bit-mask to extract a character

COLOR_PAIR(n) | Color-pair number *n*
 The following macro is the reverse of `COLOR_PAIR(n)`:

PAIR_NUMBER(attrs) | Returns the pair number associated with the `COLOR_PAIR(n)` attribute

RETURN VALUES | These routines always return 1.

ATTRIBUTES | See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO | `curs_addch(3X)`, `curs_addstr(3X)`, `curs_printw(3X)`, `curses(3X)`, `attributes(5)`

NOTES | The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

Note that `attroff()`, `wattroff()`, `attron()`, `wattron()`, `wattrset()`, `standend()`, and `standout()` may be macros.

| NAME | curs_beep, beep, flash – curses bell and screen flash routines | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lcurses [library ...] #include <curses.h> int beep(void); int flash(void);</pre> | | | | |
| DESCRIPTION | <p>The beep() and flash() routines are used to signal the terminal user. The routine beep() sounds the audible alarm on the terminal, if possible; if that is not possible, it flashes the screen (visible bell), if that is possible. The routine flash() flashes the screen, and if that is not possible, sounds the audible signal. If neither signal is possible, nothing happens. Nearly all terminals have an audible signal (bell or beep), but only some can flash the screen.</p> | | | | |
| RETURN VALUES | These routines always return OK . | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | curses(3X) , attributes(5) | | | | |
| NOTES | The header <code><curses.h></code> automatically includes the headers <code><stdio.h></code> and <code><unctrl.h></code> . | | | | |

| | |
|----------------------|---|
| NAME | curs_bkgd, bkgd, bkgdset, wbkgdset, wbkgd – curses window background manipulation routines |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> int bkgd(chtype <i>ch</i>); void bkgdset(chtype <i>ch</i>); void wbkgdset(WINDOW * <i>win</i>, chtype <i>ch</i>); int wbkgd(WINDOW * <i>win</i>, chtype <i>ch</i>);</pre> |
| DESCRIPTION | <p>The bkgdsets() and wbkgdset() routines manipulate the background of the named window. Background is a <code>chtype</code> consisting of any combination of attributes and a character. The attribute part of the background is combined (ORed) with all non-blank characters that are written into the window with waddch(). Both the character and attribute parts of the background are combined with the blank characters. The background becomes a property of the character and moves with the character through any scrolling and insert/delete line/character operations. To the extent possible on a particular terminal, the attribute part of the background is displayed as the graphic rendition of the character put on the screen.</p> <p>The bkgd() and wbkgd() routines combine the new background with every position in the window. Background is any combination of attributes and a character. Only the attribute part is used to set the background of non-blank characters, while both character and attributes are used for blank positions. To the extent possible on a particular terminal, the attribute part of the background is displayed as the graphic rendition of the character put on the screen.</p> |
| RETURN VALUES | bkgd() and wbkgd() return the integer <code>OK</code> , or a non-negative integer, if immedok() is set. See curs_outopts(3X) . |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `curs_addch(3X)`, `curs_outopts(3X)`, `curses(3X)`, `attributes(5)`

NOTES The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

Note that `bkgdset()` and `bkgd()` may be macros.

NAME curs_border, border, wborder, box, whline, wvline – create curses borders, horizontal and vertical lines

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lcurses
[
library
... ]
#include <curses.h>

int border(chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, chtype tr, chtype bl,
chtype br);

int wborder(WINDOW * win, chtype ls, chtype rs, chtype ts, chtype bs, chtype tl, chtype
bl, chtype br);

int box(WINDOW * win, chtype verch, chtype horch);

int hline(chtype ch, int n);

int whline(WINDOW * win, chtype ch, int n);

int vline(chtype ch, int n);

int wvline(WINDOW * win, chtype ch, int n);
```

DESCRIPTION

With the **border()** , **wborder()** , and **box()** routines, a border is drawn around the edges of the window. The arguments and attributes are:

| | |
|-----------|---------------------------|
| <i>ls</i> | left side of the border |
| <i>rs</i> | right side of the border |
| <i>ts</i> | top side of the border |
| <i>bs</i> | bottom side of the border |
| <i>tl</i> | top left-hand corner |
| <i>tr</i> | top right-hand corner |
| <i>bl</i> | bottom left-hand corner |
| <i>br</i> | bottom right-hand corner |

If any of these arguments is zero, then the following default values (defined in `<curses.h>`) are used respectively instead: ACS_VLINE , ACS_VLINE ,

ACS_HLINE , ACS_HLINE , ACS_ULCORNER , ACS_URCORNER ,
ACS_BLCORNER , ACS_BRCORNER .

`box(win , verch , horch)` is a shorthand for the following call:

```
wborder( win , verch , verch , horch , horch , 0 , 0 , 0 , 0 )
```

hline() and **whline()** draw a horizontal (left to right) line using *ch* starting at the current cursor position in the window. The current cursor position is not changed. The line is at most *n* characters long, or as many as fit into the window.

vline() and **wvline()** draw a vertical (top to bottom) line using *ch* starting at the current cursor position in the window. The current cursor position is not changed. The line is at most *n* characters long, or as many as fit into the window.

RETURN VALUES

All routines return the integer `OK` , or a non-negative integer if `immedok()` is set. See `curs_outopts(3X)` .

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`curs_outopts(3X)` , `curses(3X)` , `attributes(5)`

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>` .

Note that `border()` and `box()` may be macros.

| | |
|----------------------|--|
| NAME | curs_clear, erase, werase, clear, wclear, clrtoobot, wclrtoobot, clrtoeol, wclrtoeol – clear all or part of a curses window |
| SYNOPSIS | <pre> cc [flag ...] file ... -lcurses [library ...] #include <curses.h> int erase(void); int werase(WINDOW * win); int clear(void); int wclear(WINDOW * win); int clrtoobot(void); int wclrtoobot(WINDOW * win); int clrtoeol(void); int wclrtoeol(WINDOW * win); </pre> |
| DESCRIPTION | <p>The erase() and werase() routines copy blanks to every position in the window.</p> <p>The clear() and wclear() routines are like erase() and werase() , but they also call clearok() , so that the screen is cleared completely on the next call to wrefresh() for that window and repainted from scratch.</p> <p>The clrtoobot() and wclrtoobot() routines erase all lines below the cursor in the window. Also, the current line to the right of the cursor, inclusive, is erased.</p> <p>The clrtoeol() and wclrtoeol() routines erase the current line to the right of the cursor, inclusive.</p> |
| RETURN VALUES | All routines return the integer OK , or a non-negative integer if immedok() is set. See curs_outopts(3X) . |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `curs_outopts(3X)`, `curs_refresh(3X)`, `curses(3X)`, `attributes(5)`

NOTES The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

Note that `erase()`, `werase()`, `clear()`, `wclear()`, `clrtoeol()`, and `clrtoeol()` may be macros.

| | |
|--------------------|--|
| NAME | curs_color, start_color, init_pair, init_color, has_colors, can_change_color, color_content, pair_content – curses color manipulation routines |
| SYNOPSIS | <pre>cc [flag ...] file ... -lcurses [library ...] #include <curses.h> int start_color(void); int init_pair(short pair, short fg, short bg); int init_color(short color, short red, short green, short blue); bool has_colors(void); bool can_change_color(void); int color_content(short color, short * redp, short * greenp, short * bluep); int pair_content(short pair, short * fg, short * bg);</pre> |
| DESCRIPTION | |
| Overview | <p><code>curses</code> provides routines that manipulate color on color alphanumeric terminals. To use these routines <code>start_color()</code> must be called, usually right after <code>initscr()</code>. See <code>curs_initscr(3X)</code>. Colors are always used in pairs (referred to as color-pairs). A color-pair consists of a foreground color (for characters) and a background color (for the field on which the characters are displayed). A programmer initializes a color-pair with the routine <code>init_pair</code>. After it has been initialized, <code>COLOR_PAIR(n)</code>, a macro defined in <code><curses.h></code>, can be used in the same ways other video attributes can be used. If a terminal is capable of redefining colors, the programmer can use the routine <code>init_color()</code> to change the definition of a color. The routines <code>has_colors()</code> and <code>can_change_color()</code> return <code>TRUE</code> or <code>FALSE</code>, depending on whether the terminal has color capabilities and whether the programmer can change the colors. The routine <code>color_content()</code> allows a programmer to identify the amounts of red, green, and blue components in an initialized color. The routine <code>pair_content()</code> allows a programmer to find out how a given color-pair is currently defined.</p> |

Routine Descriptions

The **start_color()** routine requires no arguments. It must be called if the programmer wants to use colors, and before any other color manipulation routine is called. It is good practice to call this routine right after **initscr()**. **start_color()** initializes eight basic colors (black, red, green, yellow, blue, magenta, cyan, and white), and two global variables, `COLORS` and `COLOR_PAIRS` (respectively defining the maximum number of colors and color-pairs the terminal can support). It also restores the colors on the terminal to the values they had when the terminal was just turned on.

The **init_pair()** routine changes the definition of a color-pair. It takes three arguments: the number of the color-pair to be changed, the foreground color number, and the background color number. The value of the first argument must be between 1 and `COLOR_PAIRS - 1`. The value of the second and third arguments must be between 0 and `COLORS`. If the color-pair was previously initialized, the screen is refreshed and all occurrences of that color-pair is changed to the new definition.

The **init_color()** routine changes the definition of a color. It takes four arguments: the number of the color to be changed followed by three RGB values (for the amounts of red, green, and blue components). The value of the first argument must be between 0 and `COLORS`. (See the section `Colors` for the default color index.) Each of the last three arguments must be a value between 0 and 1000. When **init_color()** is used, all occurrences of that color on the screen immediately change to the new definition.

The **has_colors()** routine requires no arguments. It returns `TRUE` if the terminal can manipulate colors; otherwise, it returns `FALSE`. This routine facilitates writing terminal-independent programs. For example, a programmer can use it to decide whether to use color or some other video attribute.

The **can_change_color()** routine requires no arguments. It returns `TRUE` if the terminal supports colors and can change their definitions; other, it returns `FALSE`. This routine facilitates writing terminal-independent programs.

The **color_content()** routine gives users a way to find the intensity of the red, green, and blue (RGB) components in a color. It requires four arguments: the color number, and three addresses of `short` `s` for storing the information about the amounts of red, green, and blue components in the given color. The value of the first argument must be between 0 and `COLORS`. The values that are stored at the addresses pointed to by the last three arguments are between 0 (no component) and 1000 (maximum amount of component).

The **pair_content()** routine allows users to find out what colors a given color-pair consists of. It requires three arguments: the color-pair number, and two addresses of `short` `s` for storing the foreground and the background color numbers. The value of the first argument must be between 1 and

COLOR_PAIRS -1. The values that are stored at the addresses pointed to by the second and third arguments are between 0 and COLORS .

Colors

In < curses .h> the following macros are defined. These are the default colors. curses also assumes that COLOR_BLACK is the default background color for all terminals.

COLOR_BLACK
 COLOR_RED
 COLOR_GREEN
 COLOR_YELLOW
 COLOR_BLUE
 COLOR_MAGENTA
 COLOR_CYAN
 COLOR_WHITE

RETURN VALUES

All routines that return an integer return ERR upon failure and OK upon successful completion.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

curs_attr(3X) , curs_initscr(3X) , curses(3X) , attributes(5)

NOTES

The header < curses .h> automatically includes the headers < stdio .h> and < unctrl .h> .

| NAME | curs_delch, delch, wdelch, mvdelch, mvwdelch – delete character under cursor in a curses window | | | | |
|----------------------|---|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> int delch(void); int wdelch(WINDOW * win); int mvdelch(int y, int x); int mvwdelch(WINDOW * win, int y, int x);</pre> | | | | |
| DESCRIPTION | With these routines the character under the cursor in the window is deleted; all characters to the right of the cursor on the same line are moved to the left one position and the last character on the line is filled with a blank. The cursor position does not change (after moving to <i>y</i> , <i>x</i> , if specified). This does not imply use of the hardware delete character feature. | | | | |
| RETURN VALUES | All routines return the integer <code>ERR</code> upon failure and an integer value other than <code>ERR</code> upon successful completion. | | | | |
| ATTRIBUTES | See <code>attributes(5)</code> for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | <code>curses(3X)</code> , <code>attributes(5)</code> | | | | |
| NOTES | The header <code><curses.h></code> automatically includes the headers <code><stdio.h></code> and <code><unctrl.h></code> . | | | | |
| | Note that <code>delch()</code> , <code>mvdelch()</code> , and <code>mvwdelch()</code> may be macros. | | | | |

| | |
|----------------------|--|
| NAME | curs_deleteln, deleteln, wdeleteln, insdelln, winsdelln, insertln, winsertln – delete and insert lines in a curses window |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> int deleteln(void); int wdeleteln(WINDOW * win); int insdelln(int n); int winsdelln(WINDOW * win, int n); int insertln(void); int winsertln(WINDOW * win);</pre> |
| DESCRIPTION | <p>With the deleteln() and wdeleteln() routines, the line under the cursor in the window is deleted; all lines below the current line are moved up one line. The bottom line of the window is cleared. The cursor position does not change. This does not imply use of a hardware delete line feature.</p> <p>With the insdelln() and winsdelln() routines, for positive <i>n</i> , insert <i>n</i> lines into the specified window above the current line. The <i>n</i> bottom lines are lost. For negative <i>n</i> , delete <i>n</i> lines (starting with the one under the cursor), and move the remaining lines up. The bottom <i>n</i> lines are cleared. The current cursor position remains the same.</p> <p>With the insertln() and winsertln() routines, a blank line is inserted above the current line and the bottom line is lost. This does not imply use of a hardware insert line feature.</p> |
| RETURN VALUES | All routines return the integer <code>ERR</code> upon failure and an integer value other than <code>ERR</code> upon successful completion. |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `curses(3X)` , `attributes(5)`

NOTES The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>` .

Note that all but `winsdelln()` may be macros.

| | |
|--------------------|---|
| NAME | curses – CRT screen handling and optimization package |
| SYNOPSIS | cc [<i>flag...</i>] <i>file...</i> -lcurses [<i>library...</i>] #include <curses.h> |
| DESCRIPTION | <p>The <code>curses</code> library routines give the user a terminal-independent method of updating character screens with reasonable optimization.</p> <p>The <code>curses</code> package allows: overall screen, window and pad manipulation; output to windows and pads; reading terminal input; control over terminal and <code>curses</code> input and output options; environment query routines; color manipulation; use of soft label keys; terminfo access; and access to low-level <code>curses</code> routines.</p> <p>To initialize the routines, the routine <code>initscr()</code> or <code>newterm()</code> must be called before any of the other routines that deal with windows and screens are used. The routine <code>endwin()</code> must be called before exiting. To get character-at-a-time input without echoing (most interactive, screen oriented programs want this), the following sequence should be used:</p> <pre>initscr,cbreak,noecho;</pre> <p>Most programs would additionally use the sequence:</p> <pre>nonl,intrflush(stdscr,FALSE),keypad(stdscr,TRUE);</pre> <p>Before a <code>curses</code> program is run, the tab stops of the terminal should be set and its initialization strings, if defined, must be output. This can be done by executing the <code>tput init</code> command after the shell environment variable <code>TERM</code> has been exported. (See <code>terminfo(4)</code> for further details.)</p> <p>The <code>curses</code> library permits manipulation of data structures, called <i>windows</i>, which can be thought of as two-dimensional arrays of characters representing all or part of a CRT screen. A default window called <code>stdscr</code>, which is the size of the terminal screen, is supplied. Others may be created with <code>newwin(3X)</code>.</p> <p>Windows are referred to by variables declared as <code>WINDOW *</code>. These data structures are manipulated with routines described on 3X pages (whose names begin "curs_"). Among which the most basic routines are <code>move(3X)</code> and <code>addch(3X)</code>. More general versions of these routines are included with names beginning with <code>w</code>, allowing the user to specify a window. The routines not beginning with <code>w</code> affect <code>stdscr</code>.</p> <p>After using routines to manipulate a window, <code>refresh(3X)</code> is called, telling <code>curses</code> to make the user's CRT screen look like <code>stdscr</code>. The characters in a</p> |

window are actually of type `chtype`, (character and attribute data) so that other information about the character may also be stored with each character.

Special windows called *pads* may also be manipulated. These are windows which are not constrained to the size of the screen and whose contents need not be completely displayed. See `curs_pad(3X)` for more information.

In addition to drawing characters on the screen, video attributes and colors may be included, causing the characters to show up in such modes as underlined, in reverse video, or in color on terminals that support such display enhancements. Line drawing characters may be specified to be output. On input, `curses` is also able to translate arrow and function keys that transmit escape sequences into single values. The video attributes, line drawing characters, and input values use names, defined in `<curses.h>`, such as `A_REVERSE`, `ACS_HLINE`, and `KEY_LEFT`.

If the environment variables `LINES` and `COLUMNS` are set, or if the program is executing in a window environment, line and column information in the environment will override information read by *terminfo*. This would effect a program running in an AT&T 630 layer, for example, where the size of a screen is changeable.

If the environment variable `TERMINFO` is defined, any program using `curses` checks for a local terminal definition before checking in the standard place. For example, if `TERM` is set to `att4424`, then the compiled terminal definition is found in

```
/usr/share/lib/terminfo/a/att4424.
```

(The 'a' is copied from the first letter of `att4424` to avoid creation of huge directories.) However, if `TERMINFO` is set to `$HOME/myterms`, `curses` first checks

```
$HOME/myterms/a/att4424,
```

and if that fails, it then checks

```
/usr/share/lib/terminfo/a/att4424.
```

This is useful for developing experimental definitions or when write permission in `/usr/share/lib/terminfo` is not available.

The integer variables `LINES` and `COLS` are defined in `<curses.h>` and will be filled in by `initscr` with the size of the screen. The constants `TRUE` and `FALSE` have the values 1 and 0, respectively.

**International
Functions**

The `curses` routines also define the `WINDOW *` variable `curscr` which is used for certain low-level operations like clearing and redrawing a screen containing garbage. The `curscr` can be used in only a few routines.

The number of bytes and the number of columns to hold a character from the supplementary character set is locale-specific (locale category `LC_CTYPE`) and can be specified in the character class table.

For editing, operating at the character level is entirely appropriate. For screen formatting, arbitrary movement of characters on screen is not desirable.

Overwriting characters (`addch`, for example) operates on a screen level. Overwriting a character by a character that requires a different number of columns may produce *orphaned columns*. These orphaned columns are filled with background characters.

Inserting characters (`insch`, for example) operates on a character level (that is, at the character boundaries). The specified character is inserted right before the character, regardless of which column of a character the cursor points to. Before insertion, the cursor position is adjusted to the first column of the character.

As with inserting characters, deleting characters (`delch`, for example) operates on a character level (that is, at the character boundaries). The character at the cursor is deleted whichever column of the character the cursor points to. Before deletion, the cursor position is adjusted to the first column of the character.

A *multi-column* character cannot be put on the last column of a line. When such attempts are made, the last column is set to the background character. In addition, when such an operation creates orphaned columns, the orphaned columns are filled with background characters.

Overlapping and overwriting a window follows the operation of overwriting characters around its edge. The orphaned columns, if any, are handled as in the character operations.

The cursor is allowed to be placed anywhere in a window. If the insertion or deletion is made when the cursor points to the second or later column position of a character that holds multiple columns, the cursor is adjusted to the first column of the character before the insertion or deletion.

**Routine and
Argument Names**

Many `curses` routines have two or more versions. The routines prefixed with `w` require a window argument. The routines prefixed with `p` require a pad argument. Those without a prefix generally use `stdscr`.

The routines prefixed with `mv` require an `x` and `y` coordinate to move to before performing the appropriate action. The `mv` routines imply a call to `move(3X)` before the call to the other routine. The coordinate `y` always refers to the row

(of the window), and *x* always refers to the column. The upper left-hand corner is always (0,0), not (1,1).

The routines prefixed with `mvw` take both a window argument and *x* and *y* coordinates. The window argument is always specified before the coordinates.

In each case, *win* is the window affected, and *pad* is the pad affected; *win* and *pad* are always pointers to type `WINDOW`

Option setting routines require a Boolean flag *bf* with the value `TRUE` or `FALSE`; *bf* is always of type `bool`. The variables *ch* and *attrs* below are always of type `chtype`. The types `WINDOW`, `SCREEN`, `bool`, and `chtype` are defined in `<curses.h>`. The type `TERMINAL` is defined in `<term.h>`. All other arguments are integers.

Routine Name Index

The following table lists each `curses` routine and the name of the manual page on which it is described.

| curses Routine Name | Manual Page Name |
|----------------------------|---------------------------------|
| addch | <code>curs_addch(3X)</code> |
| addchnstr | <code>curs_addchstr(3X)</code> |
| addchstr | <code>curs_addchstr(3X)</code> |
| addnstr | <code>curs_addstr(3X)</code> |
| addnwstr | <code>curs_addwstr(3X)</code> |
| addstr | <code>curs_addstr(3X)</code> |
| addwch | <code>curs_addwch(3X)</code> |
| addwchnstr | <code>curs_addwchstr(3X)</code> |
| addwchstr | <code>curs_addwchstr(3X)</code> |
| addwstr | <code>curs_addwstr(3X)</code> |
| adjcurspos | <code>curs_alecompat(3X)</code> |
| attroff | <code>curs_attr(3X)</code> |
| attron | <code>curs_attr(3X)</code> |
| attrset | <code>curs_attr(3X)</code> |
| baudrate | <code>curs_termattrs(3X)</code> |
| beep | <code>curs_beep(3X)</code> |
| bkgd | <code>curs_bkgd(3X)</code> |

| | |
|-------------------------|--------------------------|
| bkgdset | curs_bkgd(3X) |
| border | curs_border(3X) |
| box | curs_border(3X) |
| can_change_color | curs_color(3X) |
| cbreak | curs_inopts(3X) |
| clear | curs_clear(3X) |
| clearok | curs_outopts(3X) |
| clrtoebot | curs_clear(3X) |
| clrtoeol | curs_clear(3X) |
| color_content | curs_color(3X) |
| copywin | curs_overlay(3X) |
| curs_set | curs_kernel(3X) |
| def_prog_mode | curs_kernel(3X) |
| def_shell_mode | curs_kernel(3X) |
| del_curterm | curs_terminfo(3X) |
| delay_output | curs_util(3X) |
| delch | curs_delch(3X) |
| deleteln | curs_deleteln(3X) |
| delscreen | curs_initscr(3X) |
| delwin | curs_window(3X) |
| derwin | curs_window(3X) |
| doupdate | curs_refresh(3X) |
| dupwin | curs_window(3X) |
| echo | curs_inopts(3X) |
| echochar | curs_addch(3X) |
| echowchar | curs_addwch(3X) |
| endwin | curs_initscr(3X) |
| erase | curs_clear(3X) |

| | |
|-------------------|---------------------------------|
| erasechar | <code>curs_termattrs(3X)</code> |
| filter | <code>curs_util(3X)</code> |
| flash | <code>curs_beep(3X)</code> |
| flushinp | <code>curs_util(3X)</code> |
| getbegyx | <code>curs_getyx(3X)</code> |
| getch | <code>curs_getch(3X)</code> |
| getmaxyx | <code>curs_getyx(3X)</code> |
| getnwstr | <code>curs_getwstr(3X)</code> |
| getparyx | <code>curs_getyx(3X)</code> |
| getstr | <code>curs_getstr(3X)</code> |
| getsyx | <code>curs_kernel(3X)</code> |
| getwch | <code>curs_getwch(3X)</code> |
| getwin | <code>curs_util(3X)</code> |
| getwstr | <code>curs_getwstr(3X)</code> |
| getyx | <code>curs_getyx(3X)</code> |
| halfdelay | <code>curs_inopts(3X)</code> |
| has_colors | <code>curs_color(3X)</code> |
| has_ic | <code>curs_termattrs(3X)</code> |
| has_il | <code>curs_termattrs(3X)</code> |
| idcok | <code>curs_outopts(3X)</code> |
| idlok | <code>curs_outopts(3X)</code> |
| immedok | <code>curs_outopts(3X)</code> |
| inch | <code>curs_inch(3X)</code> |
| inchnstr | <code>curs_inchstr(3X)</code> |
| inchstr | <code>curs_inchstr(3X)</code> |
| init_color | <code>curs_color(3X)</code> |
| init_pair | <code>curs_color(3X)</code> |

| | |
|-----------------------|----------------------------|
| initscr | curs_initscr(3X) |
| innstr | curs_instr(3X) |
| innwstr | curs_inwstr(3X) |
| insch | curs_insch(3X) |
| insdelln | curs_deleteln(3X) |
| insertln | curs_deleteln(3X) |
| insnstr | curs_insnstr(3X) |
| insnwstr | curs_inswstr(3X) |
| insstr | curs_insnstr(3X) |
| instr | curs_instr(3X) |
| inswch | curs_inswch(3X) |
| inswstr | curs_inswstr(3X) |
| intrflush | curs_inopts(3X) |
| inwch | curs_inwch(3X) |
| inwchnstr | curs_inwchstr(3X) |
| inwchstr | curs_inwchstr(3X) |
| inwstr | curs_inwstr(3X) |
| is_linetouched | curs_touch(3X) |
| is_wintouched | curs_touch(3X) |
| isendwin | curs_initscr(3X) |
| keyname | curs_util(3X) |
| keypad | curs_inopts(3X) |
| killchar | curs_termattrs(3X) |
| leaveok | curs_outopts(3X) |
| longname | curs_termattrs(3X) |
| meta | curs_inopts(3X) |
| move | curs_move(3X) |

| | |
|---------------------|---------------------------|
| movenextch | curs_alecompat(3X) |
| moveprevch | curs_alecompat(3X) |
| mvaddch | curs_addch(3X) |
| mvaddchnstr | curs_addchstr(3X) |
| mvaddchstr | curs_addchstr(3X) |
| mvaddnstr | curs_addstr(3X) |
| mvaddnwstr | curs_addwstr(3X) |
| mvaddstr | curs_addstr(3X) |
| mvaddwch | curs_addwch(3X) |
| mvaddwchnstr | curs_addwchstr(3X) |
| mvaddwchstr | curs_addwchstr(3X) |
| mvaddwstr | curs_addwstr(3X) |
| mvcur | curs_terminfo(3X) |
| mvdelch | curs_delch(3X) |
| mvderwin | curs_window(3X) |
| mvgetch | curs_getch(3X) |
| mvgetnwstr | curs_getwstr(3X) |
| mvgetstr | curs_getstr(3X) |
| mvgetwch | curs_getwch(3X) |
| mvgetwstr | curs_getwstr(3X) |
| mvinch | curs_inch(3X) |
| mvinchnstr | curs_inchstr(3X) |
| mvinchstr | curs_inchstr(3X) |
| mvinnstr | curs_instr(3X) |
| mvinnwstr | curs_inwstr(3X) |
| mvinsch | curs_insch(3X) |
| mvinsnstr | curs_insstr(3X) |
| mvinsnwstr | curs_inswstr(3X) |

| | |
|----------------------|---------------------------|
| mvinsstr | curs_insstr(3X) |
| mvinstr | curs_instr(3X) |
| mvinswch | curs_inswch(3X) |
| mvinswstr | curs_inswstr(3X) |
| mvinwch | curs_inwch(3X) |
| mvinwchnstr | curs_inwchstr(3X) |
| mvinwchstr | curs_inwchstr(3X) |
| mvinwstr | curs_inwstr(3X) |
| mvprintw | curs_printw(3X) |
| mvscanw | curs_scanw(3X) |
| mvwaddch | curs_addch(3X) |
| mvwaddchnstr | curs_addchstr(3X) |
| mvwaddchstr | curs_addchstr(3X) |
| mvwaddnstr | curs_addstr(3X) |
| mvwaddnwstr | curs_addwstr(3X) |
| mvwaddstr | curs_addstr(3X) |
| mvwaddwch | curs_addwch(3X) |
| mvwaddwchnstr | curs_addwchstr(3X) |
| mvwaddwchstr | curs_addwchstr(3X) |
| mvwaddwstr | curs_addwstr(3X) |
| mvwdelch | curs_delch(3X) |
| mvwgetch | curs_getch(3X) |
| mvwgetnwstr | curs_getwstr(3X) |
| mvwgetstr | curs_getstr(3X) |
| mvwgetwch | curs_getwch(3X) |
| mvwgetwstr | curs_getwstr(3X) |
| mvwin | curs_window(3X) |
| mvwinch | curs_inch(3X) |

| | |
|---------------------|---------------------------|
| mvwinchnstr | curs_inchstr(3X) |
| mvwinchstr | curs_inchstr(3X) |
| mvwinnstr | curs_instr(3X) |
| mvwinnwstr | curs_inwstr(3X) |
| mvwinsch | curs_insch(3X) |
| mvwinsnstr | curs_insstr(3X) |
| mvwinsstr | curs_insstr(3X) |
| mvwinstr | curs_instr(3X) |
| mvwinswch | curs_inswch(3X) |
| mvwinswstr | curs_inswstr(3X) |
| mvwinwch | curs_inwch(3X) |
| mvwinwchnstr | curs_inwchstr(3X) |
| mvwinwchstr | curs_inwchstr(3X) |
| mvwinwstr | curs_inwstr(3X) |
| mvwprintw | curs_printw(3X) |
| mvwscanw | curs_scanw(3X) |
| napms | curs_kernel(3X) |
| newpad | curs_pad(3X) |
| newterm | curs_initscr(3X) |
| newwin | curs_window(3X) |
| nl | curs_outopts(3X) |
| nocbreak | curs_inopts(3X) |
| nodelay | curs_inopts(3X) |
| noecho | curs_inopts(3X) |
| nonl | curs_outopts(3X) |
| noqiflush | curs_inopts(3X) |
| noraw | curs_inopts(3X) |

| | |
|-------------------------|--------------------------|
| notimeout | curs_inopts(3X) |
| overlay | curs_overlay(3X) |
| overwrite | curs_overlay(3X) |
| pair_content | curs_color(3X) |
| pechochar | curs_pad(3X) |
| pechowchar | curs_pad(3X) |
| pnoutrefresh | curs_pad(3X) |
| prefresh | curs_pad(3X) |
| printw | curs_printw(3X) |
| putp | curs_terminfo(3X) |
| putwin | curs_util(3X) |
| qiflush | curs_inopts(3X) |
| raw | curs_inopts(3X) |
| redrawwin | curs_refresh(3X) |
| refresh | curs_refresh(3X) |
| reset_prog_mode | curs_kernel(3X) |
| reset_shell_mode | curs_kernel(3X) |
| resetty | curs_kernel(3X) |
| restartterm | curs_terminfo(3X) |
| ripline | curs_kernel(3X) |
| savetty | curs_kernel(3X) |
| scanw | curs_scanw(3X) |
| scr_dump | curs_scr_dump(3X) |
| scr_init | curs_scr_dump(3X) |
| scr_restore | curs_scr_dump(3X) |
| scr_set | curs_scr_dump(3X) |

| | |
|------------------------|---------------------------------|
| scroll | <code>curs_scroll(3X)</code> |
| scrollok | <code>curs_outopts(3X)</code> |
| set_curterm | <code>curs_terminfo(3X)</code> |
| set_term | <code>curs_initscr(3X)</code> |
| setscreg | <code>curs_outopts(3X)</code> |
| setsyx | <code>curs_kernel(3X)</code> |
| setterm | <code>curs_terminfo(3X)</code> |
| setupterm | <code>curs_terminfo(3X)</code> |
| slk_attroff | <code>curs_slk(3X)</code> |
| slk_atron | <code>curs_slk(3X)</code> |
| slk_attrset | <code>curs_slk(3X)</code> |
| slk_clear | <code>curs_slk(3X)</code> |
| slk_init | <code>curs_slk(3X)</code> |
| slk_label | <code>curs_slk(3X)</code> |
| slk_noutrefresh | <code>curs_slk(3X)</code> |
| slk_refresh | <code>curs_slk(3X)</code> |
| slk_restore | <code>curs_slk(3X)</code> |
| slk_set | <code>curs_slk(3X)</code> |
| slk_touch | <code>curs_slk(3X)</code> |
| srcl | <code>curs_scroll(3X)</code> |
| standend | <code>curs_attr(3X)</code> |
| standout | <code>curs_attr(3X)</code> |
| start_color | <code>curs_color(3X)</code> |
| subpad | <code>curs_pad(3X)</code> |
| subwin | <code>curs_window(3X)</code> |
| syncok | <code>curs_window(3X)</code> |
| termattrs | <code>curs_termattrs(3X)</code> |

| | |
|-------------------|----------------------------|
| termname | curs_termattrs(3X) |
| tgetent | curs_termcap(3X) |
| tgetflag | curs_termcap(3X) |
| tgetnum | curs_termcap(3X) |
| tgetstr | curs_termcap(3X) |
| tgoto | curs_termcap(3X) |
| tigetflag | curs_terminfo(3X) |
| tigetnum | curs_terminfo(3X) |
| tigetstr | curs_terminfo(3X) |
| timeout | curs_inopts(3X) |
| touchline | curs_touch(3X) |
| touchwin | curs_touch(3X) |
| tparam | curs_terminfo(3X) |
| tputs | curs_terminfo(3X) |
| typeahead | curs_inopts(3X) |
| unctrl | curs_util(3X) |
| ungetch | curs_getch(3X) |
| ungetwch | curs_getwch(3X) |
| untouchwin | curs_touch(3X) |
| use_env | curs_util(3X) |
| vidattr | curs_terminfo(3X) |
| vidputs | curs_terminfo(3X) |
| vwprintw | curs_printw(3X) |
| vwscanw | curs_scanw(3X) |
| waddch | curs_addch(3X) |
| waddchnstr | curs_addchstr(3X) |

| | |
|--------------------|---------------------------------|
| waddchstr | <code>curs_addchstr(3X)</code> |
| waddnstr | <code>curs_addstr(3X)</code> |
| waddnwstr | <code>curs_addwstr(3X)</code> |
| waddstr | <code>curs_addstr(3X)</code> |
| waddwch | <code>curs_addwch(3X)</code> |
| waddwchnstr | <code>curs_addwchstr(3X)</code> |
| waddwchstr | <code>curs_addwchstr(3X)</code> |
| waddwstr | <code>curs_addwstr(3X)</code> |
| wadjcurspos | <code>curs_alecompat(3X)</code> |
| wattroff | <code>curs_attr(3X)</code> |
| watron | <code>curs_attr(3X)</code> |
| wattrset | <code>curs_attr(3X)</code> |
| wbkgd | <code>curs_bkgd(3X)</code> |
| wbkgdset | <code>curs_bkgd(3X)</code> |
| wborder | <code>curs_border(3X)</code> |
| wclear | <code>curs_clear(3X)</code> |
| wclrtoobot | <code>curs_clear(3X)</code> |
| wclrtoeol | <code>curs_clear(3X)</code> |
| wcursyncup | <code>curs_window(3X)</code> |
| wdelch | <code>curs_delch(3X)</code> |
| wdeleteln | <code>curs_deleteln(3X)</code> |
| wechochar | <code>curs_addch(3X)</code> |
| wechowchar | <code>curs_addwch(3X)</code> |
| werase | <code>curs_clear(3X)</code> |
| wgetch | <code>curs_getch(3X)</code> |
| wgetnstr | <code>curs_getstr(3X)</code> |
| wgetnwstr | <code>curs_getwstr(3X)</code> |

| | |
|---------------------|---------------------------|
| wgetstr | curs_getstr(3X) |
| wgetwch | curs_getwch(3X) |
| wgetwstr | curs_getwstr(3X) |
| whline | curs_border(3X) |
| winch | curs_inch(3X) |
| winchnstr | curs_inchstr(3X) |
| winchstr | curs_inchstr(3X) |
| winnstr | curs_instr(3X) |
| winnwstr | curs_inwstr(3X) |
| winsch | curs_insch(3X) |
| winsdelln | curs_deleteln(3X) |
| winsertln | curs_deleteln(3X) |
| winsnstr | curs_insstr(3X) |
| winsnwstr | curs_inswstr(3X) |
| winsstr | curs_insstr(3X) |
| winstr | curs_instr(3X) |
| winswch | curs_inswch(3X) |
| winswstr | curs_inswstr(3X) |
| winwch | curs_inwch(3X) |
| winwchnstr | curs_inwchstr(3X) |
| winwchstr | curs_inwchstr(3X) |
| winwstr | curs_inwstr(3X) |
| wmove | curs_move(3X) |
| wmovenextch | curs_alecompat(3X) |
| wmoveprevch | curs_alecompat(3X) |
| wnoutrefresh | curs_refresh(3X) |
| wprintw | curs_printw(3X) |

| | |
|------------------|-------------------------------|
| wredrawln | <code>curs_refresh(3X)</code> |
| wrefresh | <code>curs_refresh(3X)</code> |
| wscanw | <code>curs_scanw(3X)</code> |
| wscrl | <code>curs_scroll(3X)</code> |
| wsetscreg | <code>curs_outopts(3X)</code> |
| wstandend | <code>curs_attr(3X)</code> |
| wstandout | <code>curs_attr(3X)</code> |
| wsyncdown | <code>curs_window(3X)</code> |
| wsyncup | <code>curs_window(3X)</code> |
| wtimeout | <code>curs_inopts(3X)</code> |
| wtouchln | <code>curs_touch(3X)</code> |
| wvline | <code>curs_border(3X)</code> |

RETURN VALUES

Routines that return an integer return `ERR` upon failure and an integer value other than `ERR` upon successful completion, unless otherwise noted in the routine descriptions.

All macros return the value of the `w` version, except `setscreg()`, `wsetscreg()`, `getyx()`, `getbegyx()`, and `getmaxyx()`. The return values of `setscreg()`, `wsetscreg()`, `getyx()`, `getbegyx()`, and `getmaxyx()` are undefined (that is, these should not be used as the right-hand side of assignment statements).

Routines that return pointers return `NULL` on error.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`terminfo(4)`, `attributes(5)` and 3X pages whose names begin with “`curs_`” for detailed routine descriptions.

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

| | |
|--------------------|--|
| NAME | curses – introduction and overview of X/Open Curses |
| DESCRIPTION | <p>The X/Open Curses screen management package conforms fully with Issue 4 of the X/Open Extended Curses specification. It provides a set of internationalized functions and macros for creating and modifying input and output to a terminal screen. This includes functions for creating windows, highlighting text, writing to the screen, reading from user input, and moving the cursor. X/Open Curses is designed to optimize screen update activities.</p> <p>X/Open Curses is a terminal-independent package, providing a common user interface to a variety of terminal types. Its portability is facilitated by the Terminfo database which contains a compiled definition of each terminal type. By referring to the database information X/Open Curses gains access to low-level details about individual terminals.</p> <p>X/Open Curses tailors its activities to the terminal type specified by the TERM environment variable. The TERM environment variable may be set in the Korn Shell (see <code>ksh(1)</code>) by typing:</p> <pre>export TERM=<i>terminal_name</i></pre> <p>To set environment variables using other command line interfaces or shells, see the <code>environ(5)</code> manual page.</p> <p>Three additional environment variables are useful, and can be set in the Korn Shell:</p> <ol style="list-style-type: none">1. If you have an alternate Terminfo database containing terminal types that are not available in the system default database <code>/usr/lib/terminfo</code>, you can specify the <code>TERMINFO</code> environment variable to point to this alternate database: <pre>export TERMINFO=<i>path</i></pre> <p>This <i>path</i> specifies the location of the alternate compiled Terminfo database whose structure consists of directory names 0 to 9 and a to z (which represent the first letter of the compiled terminal definition file name).</p> <p>The alternate database specified by <code>TERMINFO</code> is examined before the system default database. If the terminal type specified by <code>TERM</code> cannot be found in either database, the default terminal type <code>dumb</code> is assumed.</p> |

2. To specify a window width smaller than your screen width (for example, in situations where your communications line is slow), set the `COLUMNS` environment variable to the number of vertical columns you want between the left and right margins:

```
export COLUMNS=number
```

The *number* of columns may be set to a number smaller than the screen size; however, if set larger than the screen or window width, the results are undefined.

The value set using this environment variable takes precedence over the value normally used for the terminal.

3. To specify a window height smaller than your current screen height (for example, in situations where your communications line is slow), override the `LINES` environment variable by setting it to a smaller number of horizontal lines:

```
export LINES=number
```

The *number* of lines may be set to a number smaller than the screen height; however, if set larger than the screen or window height, the results are undefined.

The value set using this environment variable takes precedence over the value normally used for the terminal.

Data Types

X/Open Curses defines the following data types:

| | |
|----------------------|--|
| <code>attr_t</code> | an integral type that holds an OR-ed set of attributes. The attributes acceptable are those which begin with the <code>WA_</code> prefix (see <code>Attributes</code> , <code>Color Pairs</code> , and <code>Renditions</code>). |
| <code>bool</code> | Boolean data type. |
| <code>cchar_t</code> | a type that refers to a string consisting of a spacing wide character, up to 5 non-spacing wide characters, and zero or more attributes of any type (see <code>Attributes</code> , <code>Color Pairs</code> , and <code>Renditions</code>). A null <code>cchar_t</code> object terminates arrays of <code>cchar_t</code> objects. |
| <code>chtype</code> | an integral type whose values are formed by OR-ing an "unsigned char" with a color pair (see |

| | |
|-----------------------|--|
| | Attributes, Color Pairs, and Renditions) and with zero or more attributes. The attributes acceptable are those which begin with the <code>A_</code> prefix and <code>COLOR_PAIR(3XC)</code> (see Attributes, Color Pairs, and Renditions). |
| <code>SCREEN</code> | an opaque data type associated with a terminal's display screen. |
| <code>TERMINAL</code> | an opaque data type associated with a terminal. It contains information about the terminal's capabilities (as defined by <code>terminfo</code>), the terminal modes, and current state of input/output operations. |
| <code>wchar_t</code> | an integral data type whose values represent wide characters. |
| <code>WINDOW</code> | an opaque data type associated with a window. |

Screens, Windows, and Terminals

The X/Open Curses documentation refers at various points to screens, windows (also subwindows, derived windows, and pads), and terminals. The following list defines each of these terms.

Screen A screen is a terminal's physical output device. The `SCREEN` data type is associated with a terminal.

Window Window objects are two-dimensional arrays of characters and their renditions. X/Open Curses provides `stdscr`, a default window which is the size of the terminal screen. You can use the `newwin(3XC)` function to create others.

To refer to a window, use a variable declared as `WINDOW *`. X/Open Curses includes both functions that modify `stdscr`, and more general versions that let you specify a window.

There are three sub-types of windows:

Subwindow a window which has been created within another window (the parent window) and whose position has been specified with absolute screen coordinates. The `derwin(3XC)` and `subwin(3XC)` functions can be used to create subwindows.

Derived Window a subwindow whose position is defined relative to the parent window's coordinates rather than in absolute terms.

Pad a special type of window that can be larger than the screen. For more information, see the `newpad(3XC)` man page.

Attributes, Color Pairs, and Renditions**Terminal**

A terminal is the input and output device which character-based applications use to interact with the user. The `TERMINAL` data type is associated with such a device.

A character's rendition consists of its attributes (such as underlining or reverse video) and its color pair (the foreground and background colors). When using `waddstr(3XC)`, `waddchstr(3XC)`, `wprintw(3XC)`, `winsch(3XC)`, and so on, the window's rendition is combined with that character's renditions. The window rendition is the attributes and color set using the `attroff(3XC)` and `attr_off(3XC)` sets of functions. The window's background character and rendition are set with the `bkgdset(3XC)` and `bkgrndset(3XC)` sets of functions.

When spaces are written to the screen, the background character and window rendition replace the space. For example, if the background rendition and character is `A_UNDERLINE | '*'`, text written to the window appears underlined and the spaces appear as underlined asterisks.

Each character written retains the rendition that it has obtained. This allows the character to be copied "as is" to or from a window with the `addchstr(3XC)` or `inch(3XC)` functions.

You can specify attributes using the constants listed in the tables provided in this man page. The following constants modify objects of type `chtype`:

A_ Constant Values for Highlighting Attributes

| Constant | Description |
|---------------------------|-------------------------|
| <code>A_ALTCHARSET</code> | Alternate character set |
| <code>A_ATTRIBUTES</code> | Attribute mask |
| <code>A_BLINK</code> | Blinking |
| <code>A_BOLD</code> | Bold |
| <code>A_CHARTEXT</code> | Character mask |
| <code>A_COLOR</code> | Color mask |
| <code>A_DIM</code> | Dim |
| <code>A_INVIS</code> | Invisible |
| <code>A_NORMAL</code> | Disable attributes |
| <code>A_PROTECT</code> | No display |
| <code>A_REVERSE</code> | Reverse video |

| Constant | Description |
|-------------|---------------------------------|
| A_STANDOUT | Highlights specific to terminal |
| A_UNDERLINE | Underline |

The following constants modify objects of type `attr_t`:

WA_ Constant Values for Highlighting Attributes

| Constant | Description |
|---------------|---------------------------------|
| WA_ALTCHARSET | Alternate character set |
| WA_ATTRIBUTES | Attribute mask |
| WA_BLINK | Blinking |
| WA_BOLD | Bold |
| WA_DIM | Dim |
| WA_HORIZONTAL | Horizontal highlight |
| WA_INVIS | Invisible |
| WA_LEFT | Left highlist |
| WA_LOW | Low highlist |
| WA_PROTECT | No display |
| WA_REVERSE | Reverse video |
| WA_RIGHT | Right highlight |
| WA_STANDOUT | Highlights specific to terminal |
| WA_TOP | Top highlight |
| WA_UNDERLINE | Underline |
| WA_VERTICAL | Vertical highlight |

Colors always appear in pairs; the foreground color of the character itself and the background color of the field on which it is displayed. The following color macros are defined:

Color Macros

| Macro | Description |
|-------------|-------------|
| COLOR_BLACK | Black |
| COLOR_BLUE | Blue |
| COLOR_GREEN | Green |

| Macro | Description |
|---------------|-------------|
| COLOR_CYAN | Cyan |
| COLOR_RED | Red |
| COLOR_MAGENTA | Magenta |
| COLOR_YELLOW | Yellow |
| COLOR_WHITE | White |

Together, a character's attributes and its color pair form the character's rendition. A character's rendition moves with the character during any scrolling or insert/delete operations. If your terminal lacks support for the specified rendition, X/Open Curses may substitute a different rendition.

The `COLOR_PAIR(3XC)` function modifies a `chtype` object. The `PAIR_NUMBER(3XC)` function extracts the color pair from a `chtype` object.

The following functions modify a window's color:

Functions for Modifying a Window's Color

| Function | Description |
|--|--------------------------------|
| <code>attr_set()</code> , <code>wattr_set()</code> | Change the window's rendition. |
| <code>color_set()</code> , <code>wcolor_set()</code> | Set the window's color |

Non-Spacing Characters

When the `wcwidth(3C)` function returns a width of zero for a character, that character is called a non-spacing character. Non-spacing characters can be written to a window. Each non-spacing character is associated with a spacing character (that is, one which does not have a width of zero) and modifies that character. You cannot address a non-spacing character directly. Whenever you perform an X/Open Curses operation on the associated character, you are implicitly addressing the non-spacing character.

Non-spacing characters do not have a rendition. For functions that use wide characters and a rendition, X/Open Curses ignores any rendition specified for non-spacing characters. Multicolumn characters have one rendition that applies to all columns spanned.

Complex Characters

The `cchar_t` data type represents a complex character. A complex character may contain a spacing character, its associated non-spacing characters, and its rendition. This implementation of complex characters supports up to 5 non-spacing characters for each spacing character.

Display Operations

When a `cchar_t` object representing a non-spacing complex character is written to the screen, its rendition is not used, but rather it becomes associated with the rendition of the existing character at that location. The `setcchar(3XC)` function initializes an object of type `cchar_t`. The `getcchar(3XC)` function extracts the contents of a `cchar_t` object.

In adding internationalization support to X/Open Curses, every attempt was made to minimize the number of changes to the historical CURSES package. This enables programs written to use the historical implementation of CURSES to use the internationalized version with little or no modification. The following rules apply to the internationalized X/Open Curses package:

- The cursor can be placed anywhere in the window. Window and screen origins are (0,0).
- A multicolumn character cannot be displayed in the last column, because the character would appear truncated. Instead, the background character is displayed in the last column and the multicolumn character appears at the beginning of the next line. This is called wrapping.

If the original line is the last line in the scroll region and scrolling is enabled, X/Open Curses moves the contents of each line in the region to the previous line. The first line of the region is lost. The last line of the scrolling region contains any wrapped characters. The remainder of that line is filled with the background character. If scrolling is disabled, X/Open Curses truncates any character that would extend past the last column of the screen.

- Overwrites operate on screen columns. If displaying a single-column or multicolumn character results in overwriting only a portion of a multicolumn character or characters, background characters are displayed in place of the non-overwritten portions.
- Insertions and deletions operate on whole characters. The cursor is moved to the first column of the character prior to performing the operation.

Overlapping Windows

When windows overlap, it may be necessary to overwrite only part of a multicolumn character. As mentioned earlier, the non-overwritten portions are replaced with the background character. This results in issues concerning the `overwrite(3XC)`, `overlay(3XC)`, `copywin(3XC)`, `wnoutrefresh(3XC)`, and `wrefresh(3XC)` functions.

In the upcoming examples, some characters have special meanings:

- {, [, and (represent the left halves of multicolumn characters. },], and) represent the corresponding right halves of the same multicolumn characters.
- Alphanumeric characters and periods (.) represent single-column characters.

- The number sign (#) represents the background character.

The following examples show how X/Open Curses deals with a number of issues:

1. Copying single-column characters over single-column characters.

```
copywin(s, t, 0, 1, 0, 1, 1, 3, 0)

      s          t          →   t
  abcdef  .....  .bcd..
  ghijkl  .....  .hij..
```

There are no special problems with this situation.

2. Copying multicolumn characters over single-column characters.

```
copywin(s, t, 0, 1, 0, 1, 1, 3, 0)

      s          t          →   t
  a[]def  .....  .[]d..
  gh()kl  .....  .h()..
```

There are no special problems with this situation.

3. Copying single-column characters from source overlaps multicolumn characters in target.

```
copywin(s, t, 0, 1, 0, 1, 1, 3, 0)

      s          t          →   t
  []cdef  []....  #bcd..
  ghi()l  ...().  .hij#.
```

Overwriting multicolumn characters in `t` has resulted in the # background characters being required to erase the remaining halves of the target's multicolumn characters.

4. Copy incomplete multicolumn characters from source to target.

```
copywin(s, t, 0, 1, 0, 1, 1, 3, 0)

      s          t          →      t
      []cdef     123456         []bcd56
      ghi()1     789012         7hi()2
```

The] and (halves of the multicolumn characters have been copied from the source and expanded in the target outside of the specified target region.

Consider a pop-up dialog box that contains single-column characters and a base window that contains multicolumn characters and you do the following:

```
save=dupwin(dialog); /* create backing store */
overwrite(cursor, save); /* save region to be overlaid */
wrefresh(dialog); /* display dialog */
wrefresh(save); /* restore screen image */
delwin(save); /* release backing store */
```

You can use code similar to this to implement generic **popup()** and **popdown()** routines in a variety of CURSES implementations (including BSD UNIX, and UNIX System V). In the simple case where the base window contains single-column characters only, it would correctly restore the image that appeared on the screen before the dialog box was displayed.

However, with multicolumn characters, the **overwrite()** function might save a region with incomplete multicolumn characters. The `wrefresh(dialog)` statement results in the behavior described in example 3. The behavior described in this example (that is, example 4) allows the `wrefresh(save)` statement to restore the window correctly.

5. Copying an incomplete multicolumn character to region next to screen margin (not a window edge).

Case (a)

```
copywin(s, t, 0, 1, 0, 0, 1, 2, 0)

      s          t          →      t
      []cdef     123456         #cd456
      ghijkl     789012         hij012
```

The background character (#) replaces the] character that would have been copied from the source, because it is not possible to expand the multicolumn character to its complete form.

Case (b)

```
copywin(s, t, 0, 1, 0, 3, 1, 5, 0)

      s          t          →   t
  abcdef  123456          123bcd
  ghi()l  789012          789hi#
```

This is the same as Case (a) but with the right margin.

Special Characters

Some functions assign special meanings to certain special characters:

Backspace

moves the cursor one column towards the beginning of the line. If the cursor was already at the beginning of the line, it remains there. All subsequent characters are added or inserted at this point.

Carriage Return

moves the cursor to the beginning of the current line. If the cursor was already at the beginning of the line, it remains there. All subsequent characters are added or inserted at this point.

Newline

When adding characters, X/Open Curses fills the remainder of the line with the background character (effectively truncating the newline) and scrolls the window as described earlier. All subsequent characters are inserted at the start of the new line.

When inserting characters, X/Open Curses fills the remainder of the line with the background character (effectively truncating the line), moves the cursor to the beginning of a new line, and scrolls the window as described earlier. All subsequent characters are placed at the start of the new line.

Tab

moves subsequent characters to next horizontal tab stop. Default tab stops are set at 0, 8, 16, and so on.

When adding or inserting characters, X/Open Curses inserts or adds the background character into each column until the next tab stop is

| | |
|--------------------------------|---|
| | <p>reached. If there are no remaining tab stops on the current line, wrapping and scrolling occur as described earlier.</p> |
| <p>Input Processing</p> | <p>Control Characters</p> <p>When X/Open Curses functions perform special character processing, they convert control characters to the <code>^X</code> notation, where <code>X</code> is a single-column character (uppercase, if it is a letter) and writes that notation to the window. Functions that retrieve text from the window will retrieve the converted notation not the original. X/Open Curses displays non-printable bytes, that have their high bit set, using the <code>M-X</code> meta notation where <code>X</code> is the non-printable byte with its high bit turned off.</p> <p>There are four input modes possible with X/Open Curses that affect the behavior of input functions like <code>getch(3XC)</code> and <code>getnstr(3XC)</code>.</p> <p>Line Canonical (Cooked) In line input mode, the terminal driver handles the input of line units as well as <code>SIGERASE</code> and <code>SIGKILL</code> character processing. See <code>termio(7I)</code> for more information. In this mode, the <code>getch()</code> and <code>getnstr()</code> functions will not return until a complete line has been read by the terminal driver, at which point only the requested number of bytes/characters are returned. The rest of the line unit remains unread until subsequent call to the <code>getch()</code> or <code>getnstr()</code> functions. The functions <code>nocbreak(3XC)</code> and <code>noraw(3XC)</code> are used to enter this mode. These functions are described on the <code>cbreak(3XC)</code> man page which also details which <code>termios</code> flags are enabled. Of the modes available, this one gives applications the least amount of control over input. However, it is the only input mode possible on a block mode terminal.</p> |
| | <p>cbreak Mode</p> <p>Byte/character input provides a finer degree of control. The terminal driver passes each byte read to the application without interpreting erase and kill characters. It is the application's responsibility to handle line editing. It is unknown whether the signal characters (<code>SIGINTR</code>, <code>SIGQUIT</code>, <code>SIGSUSP</code>) and flow control characters (<code>SIGSTART</code>, <code>SIGSTOP</code>) are enabled. To ensure that they are,</p> |

call the **noraw()** function first, then call the **cbreak()** function.

halfdelay Mode This is the same as the **cbreak()** mode with a timeout. The terminal driver waits for a byte to be received or for a timer to expire, in which case the **getch()** function either returns a byte or **ERR** respectively. This mode overrides timeouts set for an individual window with the **wtimeout()** function.

raw Mode This mode provides byte/character input with the most control for an application. It is similar to **cbreak()** mode, but also disables signal character processing (**SIGINTR**, **SIGSUSP**, **SIGQUIT**) and flow control processing (**SIGSTART**, **SIGSTOP**) so that the application can process them as it wants.

These modes affect all X/Open Curses input. The default input mode is inherited from the parent process when the application starts up.

A timeout similar to **halfdelay(3XC)** can be applied to individual windows (see **timeout(3XC)**). The **nodelay(3XC)** function is equivalent to setting **wtimeout(3XC)** for a window with a zero timeout (non-blocking) or infinite delay (blocking).

To handle function keys, **keypad(3XC)** must be enabled. When it is enabled, the **getch()** function returns a **KEY_** constant for a uniquely encoded key defined for that terminal. When **keypad()** is disabled, the **getch()** function returns the individual bytes composing the function key (see **getch(3XC)** and **wget_wch(3XC)**). By default, **keypad()** is disabled.

When processing function keys, once the first byte is recognized, a timer is set for each subsequent byte in the sequence. If any byte in the function key sequence is not received before the timer expires, the bytes already received are pushed into a buffer and the original first byte is returned. Subsequent X/Open Curses input would take bytes from the buffer until exhausted, after which new input from the terminal will be requested. Enabling and disabling of the function key interbyte timer is handled by the **notimeout(3XC)** function. By default, **notimeout()** is disabled (that is, the timer is used).

X/Open Curses always disables the terminal driver's echo processing. The **echo(3XC)** and **noecho(3XC)** functions control X/Open Curses software echoing. When software echoing is enabled, X/Open Curses input functions echo printable characters, control keys, and meta keys in the input window at the last cursor position. Functions keys are never echoed. When software echoing is disabled, it is the application's responsibility to handle echoing.

SEE ALSO

ksh(1), COLOR_PAIR(3XC), PAIR_NUMBER(3XC), addchstr(3XC), attr_off(3XC), attroff(3XC), bkgdset(3XC), bkgrndset(3XC), cbreak(3XC), copywin(3XC), derwin(3XC), echo(3XC), getcchar(3XC), getch(3XC), getnstr(3XC), halfdelay(3XC), inch(3XC), keypad(3XC), newpad(3XC), newwin(3XC), nocbreak(3XC), nodelay(3XC), noecho(3XC), noraw(3XC), notimeout(3XC), overlay(3XC), overwrite(3XC), setcchar(3XC), subwin(3XC), timeout(3XC), waddchstr(3XC), waddstr(3XC), wwidth(3C), wget_wch(3XC), winsch(3XC), wnoutrefresh(3XC), wprintw(3XC), wrefresh(3XC), wtimeout(3XC), termio(7I), environ(5)

| | |
|--------------------|---|
| NAME | curs_getch, getch, wgetch, mvwgetch, mvvwgetch, ungetch – get (or push back) characters from curses terminal keyboard |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> int getch(void); int wgetch(WINDOW * win); int mvwgetch(int y, int x); int mvvwgetch(WINDOW * win, int y, int x); int ungetch(int ch); </pre> |
| DESCRIPTION | <p>With the getch() , wgetch() , mvwgetch() , and mvvwgetch() routines a character is read from the terminal associated with the window. In no-delay mode, if no input is waiting, the value <code>ERR</code> is returned. In delay mode, the program waits until the system passes text through to the program. Depending on the setting of cbreak() , this is after one character (cbreak mode), or after the first newline (nocbreak mode). In half-delay mode, the program waits until a character is typed or the specified timeout has been reached. Unless noecho() has been set, the character will also be echoed into the designated window.</p> <p>If the window is not a pad, and it has been moved or modified since the last call to wrefresh() , wrefresh() will be called before another character is read.</p> <p>If keypad() is <code>TRUE</code> , and a function key is pressed, the token for that function key is returned instead of the raw characters. Possible function keys are defined in <code><curses.h></code> with integers beginning with <code>0401</code> , whose names begin with <code>KEY_</code> . If a character that could be the beginning of a function key (such as escape) is received, <code>curses</code> sets a timer. If the remainder of the sequence does not come in within the designated time, the character is passed through; otherwise, the function key value is returned. For this reason, many terminals experience a delay between the time a user presses the escape key and the escape is returned to the program. Since tokens returned by these routines are outside the ASCII range, they are not printable.</p> |

Function Keys

The **ungetch()** routine places *ch* back onto the input queue to be returned by the next call to **wgetch()** .

The following function keys, defined in `<curses.h>` , might be returned by **getch()** if **keypad()** has been enabled. Note that not all of these may be supported on a particular terminal if the terminal does not transmit a unique code when the key is pressed or if the definition for the key is not present in the *terminfo* database.

| <i>Name</i> | <i>Key name</i> |
|-------------------|---|
| KEY_BREAK | Break key |
| KEY_DOWN | The four arrow keys ... |
| KEY_UP | |
| KEY_LEFT | |
| KEY_RIGHT | |
| KEY_HOME | Home key (upward+left arrow) |
| KEY_BACKSPACE | Backspace |
| KEY_F0 | Function keys; space for 64 keys is reserved. |
| KEY_F(<i>n</i>) | For $0 \leq n \leq 63$ |
| KEY_DL | Delete line |
| KEY_IL | Insert line |
| KEY_DC | Delete character |
| KEY_IC | Insert char or enter insert mode |
| KEY_EIC | Exit insert char mode |
| KEY_CLEAR | Clear screen |
| KEY_EOS | Clear to end of screen |
| KEY_EOL | Clear to end of line |
| KEY_SF | Scroll 1 line forward |
| KEY_SR | Scroll 1 line backward (reverse) |
| KEY_NPAGE | Next page |
| KEY_PPAGE | Previous page |
| KEY_STAB | Set tab |

| <i>Name</i> | <i>Key name</i> |
|-------------|--|
| KEY_CTAB | Clear tab |
| KEY_CATAB | Clear all tabs |
| KEY_ENTER | Enter or send |
| KEY_SRESET | Soft (partial) reset |
| KEY_RESET | Reset or hard reset |
| KEY_PRINT | Print or copy |
| KEY_LL | Home down or bottom (lower left). Keypad is arranged like this: (Row 1) A1 up A3 (Row 2) left B2 right (Row 3) C1 down C3 |
| KEY_A1 | Upper left of keypad |
| KEY_A3 | Upper right of keypad |
| KEY_B2 | Center of keypad |
| KEY_C1 | Lower left of keypad |
| KEY_C3 | Lower right of keypad |
| KEY_BTAB | Back tab key |
| KEY_BEG | Beg(inning) key |
| KEY_CANCEL | Cancel key |
| KEY_CLOSE | Close key |
| KEY_COMMAND | Cmd (command) key |
| KEY_COPY | Copy key |
| KEY_CREATE | Create key |
| KEY_END | End key |
| KEY_EXIT | Exit key |
| KEY_FIND | Find key |
| KEY_HELP | Help key |
| KEY_MARK | Mark key |
| KEY_MESSAGE | Message key |
| KEY_MOVE | Move key |
| KEY_NEXT | Next object key |
| KEY_OPEN | Open key |

| <i>Name</i> | <i>Key name</i> |
|---------------|-------------------------|
| KEY_OPTIONS | Options key |
| KEY_PREVIOUS | Previous object key |
| KEY_REDO | Redo key |
| KEY_REFERENCE | Reference key |
| KEY_REFRESH | Refresh key |
| KEY_REPLACE | Replace key |
| KEY_RESTART | Restart key |
| KEY_RESUME | Resume key |
| KEY_SAVE | Save key |
| KEY_SBEG | Shifted beginning key |
| KEY_SCANCEL | Shifted cancel key |
| KEY_SCOMMAND | Shifted command key |
| KEY_SCOPY | Shifted copy key |
| KEY_SCREATE | Shifted create key |
| KEY_SDC | Shifted delete char key |
| KEY_SDL | Shifted delete line key |
| KEY_SELECT | Select key |
| KEY_SEND | Shifted end key |
| KEY_SEOL | Shifted clear line key |
| KEY_SEXIT | Shifted exit key |
| KEY_SFIND | Shifted find key |
| KEY_SHELP | Shifted help key |
| KEY_SHOME | Shifted home key |
| KEY_SIC | Shifted input key |
| KEY_SLEFT | Shifted left arrow key |
| KEY_SMESSAGE | Shifted message key |
| KEY_SMOVE | Shifted move key |
| KEY_SNEXT | Shifted next key |
| KEY_SOPTIONS | Shifted options key |
| KEY_SPREVIOUS | Shifted prev key |

| <i>Name</i> | <i>Key name</i> |
|--------------|---------------------|
| KEY_SPRINT | Shifted print key |
| KEY_SREDO | Shifted redo key |
| KEY_SREPLACE | Shifted replace key |
| KEY_SRIGHT | Shifted right arrow |
| KEY_SRSUME | Shifted resume key |
| KEY_SSAVE | Shifted save key |
| KEY_SSUSPEND | Shifted suspend key |
| KEY_SUNDO | Shifted undo key |
| KEY_SUSPEND | Suspend key |
| KEY_UNDO | Undo key |

RETURN VALUES

All routines return the integer `ERR` upon failure. The `ungetch()` routine returns an integer value other than `ERR` upon successful completion. The other routines return the next input character or function key code upon successful completion.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`curs_inopts(3X)` , `curs_move(3X)` , `curs_refresh(3X)` , `curses(3X)` , `attributes(5)`

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>` .

Use of the escape key for a single character function is discouraged.

When using `getch()` , `wgetch()` , `mvgetch()` , or `mvwgetch()` , `nocbreak` mode (`nocbreak()`) and `echo` mode (`echo()`) should not be used at the same time. Depending on the state of the tty driver when each character is typed, the program may produce undesirable results.

Note that `getch()` , `mvgetch()` , and `mvwgetch()` may be macros.

NAME curs_getstr, getstr, wgetstr, mvwgetstr, mvwgetstr, wgetnstr – get character strings from curses terminal keyboard

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lcurses
[
library
... ]
#include <curses.h>

int getstr(char * str);

int wgetstr(WINDOW * win, char * str);

int mvwgetstr(int y, int x, char * str);

int mvwgetstr(WINDOW * win, int y, int x, char * str);

int wgetnstr(WINDOW * win, char * str, int n);
```

DESCRIPTION The effect of **getstr()** is as though a series of calls to **getch()** were made, until a newline or carriage return is received. The resulting value is placed in the area pointed to by the character pointer *str*. **wgetnstr()** reads at most *n* characters, thus preventing a possible overflow of the input buffer. The user's erase and kill characters are interpreted, as well as any special keys (such as function keys, HOME key, and CLEAR key.)

RETURN VALUES All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO **curs_getch(3X)** , **curses(3X)** , **attributes(5)**

NOTES The header **<curses.h>** automatically includes the headers **<stdio.h>** and **<unctrl.h>** .

Note that **getstr()** , **mvwgetstr()** , and **mvwgetstr()** may be macros.

| | |
|--------------------|--|
| NAME | curs_getwch, getwch, wgetwch, mvgetwch, mvwgetwch, ungetwch – get (or push back) wchar_t characters from curses terminal keyboard |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ..] #include <curses.h> int getwch(void); int wgetwch(WINDOW * win); int mvgetwch(int y, int x); int mvwgetwch(WINDOW * win, int y, int x); int ungetwch(int wch); </pre> |
| DESCRIPTION | <p>The <code>getwch()</code>, <code>wgetwch()</code>, <code>mvgetwch()</code>, and <code>mvwgetwch()</code> routines read an EUC character from the terminal associated with the window, transform it into a <code>wchar_t</code> character, and return a <code>wchar_t</code> character. In no-delay mode, if no input is waiting, the value <code>ERR</code> is returned. In delay mode, the program waits until the system passes text through to the program. Depending on the setting of <code>cbreak</code>, this is after one character (<code>cbreak</code> mode), or after the first newline (<code>nocbreak</code> mode). In half-delay mode, the program waits until a character is typed or the specified timeout has been reached. Unless <code>noecho</code> has been set, the character will also be echoed into the designated window.</p> <p>If the window is not a pad, and it has been moved or modified since the last call to <code>wrefresh(3X)</code>, <code>wrefresh</code> will be called before another character is read.</p> <p>If <code>keypad</code> is <code>TRUE</code>, and a function key is pressed, the token for that function key is returned instead of the raw characters. Possible function keys are defined in <code><curses.h></code> with integers beginning with <code>0401</code>, whose names begin with <code>KEY_</code>. If a character that could be the beginning of a function key (such as escape) is received, <code>curses(3X)</code> sets a timer. If the remainder of the sequence does not come in within the designated time, the character is passed through; otherwise, the function key value is returned. For this reason, many terminals experience a delay between the time a user presses the escape key and the escape is returned to the program.</p> |

Function Keys

The **ungetwch()** routine places `wch` back onto the input queue to be returned by the next call to **wgetwch()** .

The following function keys, defined in `<curses.h>` , might be returned by **getwch()** if `keypad` has been enabled. Note that not all of these may be supported on a particular terminal if the terminal does not transmit a unique code when the key is pressed or if the definition for the key is not present in the `terminfo(4)` database.

| <i>Name</i> | <i>Key name</i> |
|-------------------|---|
| KEY_BREAK | Break key |
| KEY_DOWN | The four arrow keys ... |
| KEY_UP | |
| KEY_LEFT | |
| KEY_RIGHT | |
| KEY_HOME | Home key (upward+left arrow) |
| KEY_BACKSPACE | Backspace |
| KEY_F0 | Function keys; space for 64 keys is reserved. |
| KEY_F(<i>n</i>) | For $0 \leq n \leq 63$ |
| KEY_DL | Delete line |
| KEY_IL | Insert line |
| KEY_DC | Delete character |
| KEY_IC | Insert char or enter insert mode |
| KEY_EIC | Exit insert char mode |
| KEY_CLEAR | Clear screen |
| KEY_EOS | Clear to end of screen |
| KEY_EOL | Clear to end of line |
| KEY_SF | Scroll 1 line forward |
| KEY_SR | Scroll 1 line backward (reverse) |
| KEY_NPAGE | Next page |
| KEY_PPAGE | Previous page |
| KEY_STAB | Set tab |

| <i>Name</i> | <i>Key name</i> |
|-------------|---|
| KEY_CTAB | Clear tab |
| KEY_CATAB | Clear all tabs |
| KEY_ENTER | Enter or send |
| KEY_SRESET | Soft (partial) reset |
| KEY_RESET | Reset or hard reset |
| KEY_PRINT | Print or copy |
| KEY_LL | Home down or bottom (lower left). Keypad is arranged like this: A1 up A3 left B2 right C1 down C3 |
| KEY_A1 | Upper left of keypad |
| KEY_A3 | Upper right of keypad |
| KEY_B2 | Center of keypad |
| KEY_C1 | Lower left of keypad |
| KEY_C3 | Lower right of keypad |
| KEY_BTAB | Back tab key |
| KEY_BEG | Beg(inning) key |
| KEY_CANCEL | Cancel key |
| KEY_CLOSE | Close key |
| KEY_COMMAND | Cmd (command) key |
| KEY_COPY | Copy key |
| KEY_CREATE | Create key |
| KEY_END | End key |
| KEY_EXIT | Exit key |
| KEY_FIND | Find key |
| KEY_HELP | Help key |
| KEY_MARK | Mark key |
| KEY_MESSAGE | Message key |
| KEY_MOVE | Move key |
| KEY_NEXT | Next object key |
| KEY_OPEN | Open key |
| KEY_OPTIONS | Options key |

| <i>Name</i> | <i>Key name</i> |
|---------------|-------------------------|
| KEY_PREVIOUS | Previous object key |
| KEY_REDO | Redo key |
| KEY_REFERENCE | Reference key |
| KEY_REFRESH | Refresh key |
| KEY_REPLACE | Replace key |
| KEY_RESTART | Restart key |
| KEY_RESUME | Resume key |
| KEY_SAVE | Save key |
| KEY_SBEG | Shifted beginning key |
| KEY_SCANCEL | Shifted cancel key |
| KEY_SCOMMAND | Shifted command key |
| KEY_SCOPY | Shifted copy key |
| KEY_SCREATE | Shifted create key |
| KEY_SDC | Shifted delete char key |
| KEY_SDL | Shifted delete line key |
| KEY_SELECT | Select key |
| KEY_SEND | Shifted end key |
| KEY_SEOL | Shifted clear line key |
| KEY_SEXIT | Shifted exit key |
| KEY_SFIND | Shifted find key |
| KEY_SHELP | Shifted help key |
| KEY_SHOME | Shifted home key |
| KEY_SIC | Shifted input key |
| KEY_SLEFT | Shifted left arrow key |
| KEY_SMESSAGE | Shifted message key |
| KEY_SMOVE | Shifted move key |
| KEY_SNEXT | Shifted next key |
| KEY_SOPTIONS | Shifted options key |
| KEY_SPREVIOUS | Shifted prev key |
| KEY_SPRINT | Shifted print key |

| <i>Name</i> | <i>Key name</i> |
|--------------|---------------------|
| KEY_SREDO | Shifted redo key |
| KEY_SREPLACE | Shifted replace key |
| KEY_SRIGHT | Shifted right arrow |
| KEY_SRSUME | Shifted resume key |
| KEY_SSAVE | Shifted save key |
| KEY_SSUSPEND | Shifted suspend key |
| KEY_SUNDO | Shifted undo key |
| KEY_SUSPEND | Suspend key |
| KEY_UNDO | Undo key |

RETURN VALUE All routines return the integer `ERR` upon failure and an integer value other than `ERR` upon successful completion.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `curses(3X)` , `curs_inopts(3X)` , `curs_move(3X)` , `wrefresh(3X)` , `terminfo(4)` , `attributes(5)`

NOTES The header file `<curses.h>` automatically includes the header files `<stdio.h>` , `<unctrl.h>` and `<widec.h>` .

Use of the escape key by a programmer for a single character function is discouraged.

When using `getwch()` , `wgetwch()` , `mvgetwch()` , or `mvwgetwch()` , `nocbreak` mode and `echo` mode should not be used at the same time. Depending on the state of the tty driver when each character is typed, the program may produce undesirable results.

Note that `getwch()` , `mvgetwch()` , and `mvwgetwch()` may be macros.

NAME curs_getwstr, getwstr, getnwstr, wgetwstr, wgetnwstr, mvgetwstr, mvgetnwstr, mvwgetwstr, mvwgetnwstr – get wchar_t character strings from curses terminal keyboard

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lcurses
[
library
.. ]
#include <curses.h>

int getwstr(wchar_t * wstr);

int getnwstr(wchar_t * wstr, int n);

int wgetwstr(WINDOW * win, wchar_t * wstr);

int wgetnwstr(WINDOW * win, wchar_t * wstr, int n);

int mvgetwstr(int y, int x, wchar_t * wstr);

int mvgetnwstr(int y, int x, wchar_t * wstr, int n);

int mvwgetwstr(WINDOW * win, int y, int x, wchar_t * wstr);

int mvwgetnwstr(WINDOW * win, int y, int x, wchar_t * wstr, int n);
```

DESCRIPTION The effect of **getwstr()** is as though a series of calls to **getwch(3X)** were made, until a newline and carriage return is received. The resulting value is placed in the area pointed to by the `wchar_t` pointer `wstr`. **getnwstr()** reads at most `n` `wchar_t` characters, thus preventing a possible overflow of the input buffer. The user's erase and kill characters are interpreted, as well as any special keys (such as function keys, HOME key, CLEAR key, etc.).

RETURN VALUE All routines return the integer `ERR` upon failure and an integer value other than `ERR` upon successful completion.

ATTRIBUTES See **attributes(5)** for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO | `curses(3X)` , `getwch(3X)` , `attributes(5)`

NOTES | The header file `<curses.h>` automatically includes the header files `<stdio.h>` , `<unctrl.h>` , and `<widec.h>` .

Note that all routines except `wgetnwstr()` may be macros.

NAME curs_getyx, getyx, getparyx, getbegyx, getmaxyx – get curses cursor and window coordinates

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lcurses
[
library
... ]
#include <curses.h>

void getyx(WINDOW * win, int y, int x);
void getparyx(WINDOW * win, int y, int x);
void getbegyx(WINDOW * win, int y, int x);
void getmaxyx(WINDOW * win, int y, int x);
```

DESCRIPTION

With the **getyx()** macro, the cursor position of the window is placed in the two integer variables *y* and *x*.

With the **getparyx()** macro, if *win* is a subwindow, the beginning coordinates of the subwindow relative to the parent window are placed into two integer variables, *y* and *x*. Otherwise, *-1* is placed into *y* and *x*.

Like **getyx()**, the **getbegyx()** and **getmaxyx()** macros store the current beginning coordinates and size of the specified window.

RETURN VALUES

The return values of these macros are undefined (that is, they should not be used as the right-hand side of assignment statements).

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO **curses(3X)**, **attributes(5)**

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

Note that all of these interfaces are macros and that “ & ” is not necessary before the variables *y* and *x* .

NAME curs_inch, inch, winch, mvinch, mvwinch – get a character and its attributes from a curses window

SYNOPSIS

```
cc
[
  flag
  ... ]
file
...
-lcurses
[
  library
  ... ]
#include <curses.h>

ctype inch(void);

ctype winch(WINDOW * win);

ctype mvinch(int y, int x);

ctype mvwinch(WINDOW * win, int y, int x);
```

DESCRIPTION With these routines, the character, of type `ctype`, at the current position in the named window is returned. If any attributes are set for that position, their values are OR-ed into the value returned. Constants defined in `<curses.h>` can be used with the logical AND (`&`) operator to extract the character or attributes alone.

Attributes

The following bit-masks may be AND-ed with characters returned by `winch()`.

- A_CHARTEXT** Bit-mask to extract character
- A_ATTRIBUTES** Bit-mask to extract attributes
- A_COLOR** Bit-mask to extract color-pair field information

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `curses(3X)`, `attributes(5)`

NOTES The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

Note that all of these routines may be macros.

NAME curs_inchstr, inchstr, inchnstr, winchstr, winchnstr, mvinchstr, mvinchnstr, mvwinchstr, mvwinchnstr – get a string of characters (and attributes) from a curses window

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lcurses
[
library
... ]
#include <curses.h>

int inchstr(chtype * chstr);

int inchnstr(chtype * chstr, int n);

int winchstr(WINDOW * win, chtype * chstr);

int winchnstr(WINDOW * win, chtype * chstr, int n);

int mvinchstr(int y, int x, chtype * chstr);

int mvinchnstr(int y, int x, chtype * chstr, int n);

int mvwinchstr(WINDOW * win, int y, int x, chtype * chstr);

int mvwinchnstr(WINDOW * win, int y, int x, chtype * chstr, int n);
```

DESCRIPTION With these routines, a string of type `chtype`, starting at the current cursor position in the named window and ending at the right margin of the window, is returned. The four functions with `n` as the last argument, return the string at most `n` characters long. Constants defined in `<curses.h>` can be used with the `&` (logical AND) operator to extract the character or the attribute alone from any position in the `chstr` (see `curs_inch(3X)`).

RETURN VALUES All routines return the integer `ERR` upon failure and an integer value other than `ERR` upon successful completion.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO | `curs_inch(3X)`, `curses(3X)`, `attributes(5)`

NOTES | The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

Note that all routines except `winchnstr()` may be macros.

| | |
|--------------------|---|
| NAME | curs_initscr, initscr, newterm, endwin, isendwin, set_term, delscreen – curses screen initialization and manipulation routines |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> WINDOW * initscr(void); int endwin(void); int isendwin(void); SCREEN * newterm(char * <i>type</i>, FILE * <i>outfd</i>, FILE * <i>infd</i>); SCREEN * set_term(SCREEN * <i>new</i>); void delscreen(SCREEN * <i>sp</i>);</pre> |
| DESCRIPTION | <p>initscr() is almost always the first routine that should be called (the exceptions are slk_init() , filter() , ripoffline() , use_env() and, for multiple-terminal applications, newterm() .) This determines the terminal type and initializes all <code>curses</code> data structures. initscr() also causes the first call to refresh() to clear the screen. If errors occur, initscr() writes an appropriate error message to standard error and exits; otherwise, a pointer is returned to stdscr() . If the program needs an indication of error conditions, newterm() should be used instead of initscr() ; initscr() should only be called once per application.</p> <p>A program that outputs to more than one terminal should use the newterm() routine for each terminal instead of initscr() . A program that needs an indication of error conditions, so it can continue to run in a line-oriented mode if the terminal cannot support a screen-oriented program, would also use this routine. The routine newterm() should be called once for each terminal. It returns a variable of type <code>SCREEN *</code> which should be saved as a reference to that terminal. The arguments are the <i>type</i> of the terminal to be used in place of <code>\$ TERM</code> , a file pointer for output to the terminal, and another file pointer for input from the terminal (if <i>type</i> is <code>NULL</code> , <code>\$ TERM</code> will be used). The program must also call endwin() for each terminal being used before exiting from <code>curses</code>. If newterm() is called more than once for the same terminal, the first terminal referred to must be the last one for which endwin() is called.</p> |

A program should always call **endwin()** before exiting or escaping from `curses` mode temporarily. This routine restores tty modes, moves the cursor to the lower left-hand corner of the screen and resets the terminal into the proper non-visual mode. Calling **refresh()** or **doupdate()** after a temporary escape causes the program to resume visual mode.

The **isendwin()** routine returns `TRUE` if **endwin()** has been called without any subsequent calls to **wrefresh()** , and `FALSE` otherwise.

The **set_term()** routine is used to switch between different terminals. The screen reference `new` becomes the new current terminal. The previous terminal is returned by the routine. This is the only routine which manipulates `SCREEN` pointers; all other routines affect only the current terminal.

The **delscreen()** routine frees storage associated with the `SCREEN` data structure. The **endwin()** routine does not do this, so **delscreen()** should be called after **endwin()** if a particular `SCREEN` is no longer needed.

RETURN VALUES

endwin() returns the integer `ERR` upon failure and `OK` upon successful completion.

Routines that return pointers always return `NULL` on error.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

curs_kernel(3X) , **curs_refresh(3X)** , **curs_slk(3X)** , **curs_util(3X)** , **curses(3X)** , **attributes(5)**

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>` .

Note that **initscr()** and **newterm()** may be macros.

| | |
|--------------------|---|
| NAME | curs_inopts, cbreak, nocbreak, echo, noecho, halfdelay, intrflush, keypad, meta, nodelay, notimeout, raw, noraw, noqiflush, qiflush, timeout, wtimeout, typeahead – curses terminal input option control routines |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> int cbreak(void); int nocbreak(void); int echo(void); int noecho(void); int halfdelay(int tenths); int intrflush(WINDOW * win, bool bf); int keypad(WINDOW * win, bool bf); int meta(WINDOW * win, bool bf); int nodelay(WINDOW * win, bool bf); int notimeout(WINDOW * win, bool bf); int raw(void); int noraw(void); void noqiflush(void); void qiflush(void); void timeout(int delay); void wtimeout(WINDOW * win, int delay); int typeahead(int fildes); </pre> |
| DESCRIPTION | The cbreak() and nocbreak() routines put the terminal into and out of cbreak() mode, respectively. In this mode, characters typed by the user are |

immediately available to the program, and erase/kill character-processing is not performed. When out of this mode, the tty driver buffers the typed characters until a newline or carriage return is typed. Interrupt and flow control characters are unaffected by this mode. Initially the terminal may or may not be in **cbreak()** mode, as the mode is inherited; therefore, a program should call **cbreak()** or **nocbreak()** explicitly. Most interactive programs using *curses* set the **cbreak()** mode.

Note that **cbreak()** overrides **raw()** . (See *curs_getch(3X)* for a discussion of how these routines interact with **echo()** and **noecho()** .)

The **echo()** and **noecho()** routines control whether characters typed by the user are echoed by **getch()** as they are typed. Echoing by the tty driver is always disabled, but initially **getch()** is in echo mode, so characters typed are echoed. Authors of most interactive programs prefer to do their own echoing in a controlled area of the screen, or not to echo at all, so they disable echoing by calling **noecho()** . (See *curs_getch(3X)* for a discussion of how these routines interact with **cbreak()** and **nocbreak()** .)

The **halfdelay()** routine is used for half-delay mode, which is similar to **cbreak()** mode in that characters typed by the user are immediately available to the program. However, after blocking for *tenths* tenths of seconds, **ERR** is returned if nothing has been typed. The value of *tenths* must be a number between 1 and 255. Use **nocbreak()** to leave half-delay mode.

If the **intrflush()** option is enabled, (*bf* is **TRUE**), when an interrupt key is pressed on the keyboard (interrupt, break, quit) all output in the tty driver queue will be flushed, giving the effect of faster response to the interrupt, but causing *curses* to have the wrong idea of what is on the screen. Disabling (*bf* is **FALSE**), the option prevents the flush. The default for the option is inherited from the tty driver settings. The window argument is ignored.

The **keypad()** option enables the keypad of the user's terminal. If enabled (*bf* is **TRUE**), the user can press a function key (such as an arrow key) and **wgetch()** returns a single value representing the function key, as in **KEY_LEFT** . If disabled (*bf* is **FALSE**), *curses* does not treat function keys specially and the program has to interpret the escape sequences itself. If the keypad in the terminal can be turned on (made to transmit) and off (made to work locally), turning on this option causes the terminal keypad to be turned on when **wgetch()** is called. The default value for keypad is false.

Initially, whether the terminal returns 7 or 8 significant bits on input depends on the control mode of the tty driver (see *termio(7I)*). To force 8 bits to be returned, invoke **meta (win , TRUE)** . To force 7 bits to be returned, invoke **meta (win , FALSE)** . The window argument, *win* , is always ignored. If the terminfo capabilities **smm** (*meta_on*) and **rmm** (*meta_off*) are defined for the

terminal, `smm` is sent to the terminal when `meta (win , TRUE)` is called and `rmm` is sent when `meta (win , FALSE)` is called.

The `nodelay()` option causes `getch()` to be a non-blocking call. If no input is ready, `getch()` returns `ERR`. If disabled (`bf` is `FALSE`), `getch()` waits until a key is pressed.

While interpreting an input escape sequence, `wgetch()` sets a timer while waiting for the next character. If `notimeout(win , TRUE)` is called, then `wgetch()` does not set a timer. The purpose of the timeout is to differentiate between sequences received from a function key and those typed by a user.

With the `raw()` and `noraw()` routines, the terminal is placed into or out of raw mode. Raw mode is similar to `cbreak()` mode, in that characters typed are immediately passed through to the user program. The differences are that in raw mode, the interrupt, quit, suspend, and flow control characters are all passed through uninterpreted, instead of generating a signal. The behavior of the `BREAK` key depends on other bits in the tty driver that are not set by `curses`.

When the `noqiflush()` routine is used, normal flush of input and output queues associated with the `INTR`, `QUIT` and `SUSP` characters will not be done (see `termio(7I)`). When `qiflush()` is called, the queues will be flushed when these control characters are read.

The `timeout()` and `wtimeout()` routines set blocking or non-blocking read for a given window. If `delay` is negative, blocking read is used (that is, waits indefinitely for input). If `delay` is zero, then non-blocking read is used (that is, read returns `ERR` if no input is waiting). If `delay` is positive, then read blocks for `delay` milliseconds, and returns `ERR` if there is still no input. Hence, these routines provide the same functionality as `nodelay()`, plus the additional capability of being able to block for only `delay` milliseconds (where `delay` is positive).

`curses` does "line-breakout optimization" by looking for typeahead periodically while updating the screen. If input is found, and it is coming from a tty, the current update is postponed until `refresh()` or `doupdate()` is called again. This allows faster response to commands typed in advance. Normally, the input FILE pointer passed to `newterm()`, or `stdin` in the case that `initscr()` was used, will be used to do this typeahead checking. The `typeahead()` routine specifies that the file descriptor `fildes` is to be used to check for typeahead instead. If `fildes` is `-1`, then no typeahead checking is done.

RETURN VALUES

All routines that return an integer return `ERR` upon failure and an integer value other than `ERR` upon successful completion, unless otherwise noted in the preceding routine descriptions.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

curs_getch(3X) , **curs_initscr(3X)** , **curses(3X)** , **attributes(5)** , **termio(7I)**

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>` .

Note that **echo()** , **noecho()** , **halfdelay()** , **intrflush()** , **meta()** , **nodelay()** , **notimeout()** , **noqiflush()** , **qiflush()** , **timeout()** , and **wtimeout()** may be macros.

| NAME | curs_insch, insch, wünsch, mvünsch, mvwünsch – insert a character before the character under the cursor in a curses window | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> int insch(chtype <i>ch</i>); int wünsch(WINDOW * <i>win</i>, chtype <i>ch</i>); int mvünsch(int <i>y</i>, int <i>x</i>, chtype <i>ch</i>); int mvwünsch(WINDOW * <i>win</i>, int <i>y</i>, int <i>x</i>, chtype <i>ch</i>); </pre> | | | | |
| DESCRIPTION | <p>With these routines, the character <i>ch</i> is inserted before the character under the cursor. All characters to the right of the cursor are moved one space to the right, with the possibility of the rightmost character on the line being lost. The cursor position does not change (after moving to <i>y</i>, <i>x</i>, if specified). (This does not imply use of the hardware insert character feature.)</p> | | | | |
| RETURN VALUES | <p>All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | <p>curses(3X), attributes(5)</p> | | | | |
| NOTES | <p>The header <curses.h> automatically includes the headers <stdio.h> and <unctrl.h> .</p> <p>Note that insch(), mvünsch(), and mvwünsch() may be macros.</p> | | | | |

| | |
|--------------------|--|
| NAME | curs_insstr, insstr, insnstr, winsstr, winsnstr, mvinsstr, mvinsnstr, mvwinsstr, mvwinsnstr – insert string before character under the cursor in a curses window |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> int insstr(char * <i>str</i>); int insnstr(char * <i>str</i>, int <i>n</i>); int winsstr(WINDOW * <i>win</i>, char * <i>str</i>); int winsnstr(WINDOW * <i>win</i>, char * <i>str</i>, int <i>n</i>); int mvinsstr(int <i>y</i>, int <i>x</i>, char * <i>str</i>); int mvinsnstr(int <i>y</i>, int <i>x</i>, char * <i>str</i>, int <i>n</i>); int mvwinsstr(WINDOW * <i>win</i>, int <i>y</i>, int <i>x</i>, char * <i>str</i>); int mvwinsnstr(WINDOW * <i>win</i>, int <i>y</i>, int <i>x</i>, char * <i>str</i>, int <i>n</i>); </pre> |
| DESCRIPTION | <p>With these routines, a character string (as many characters as will fit on the line) is inserted before the character under the cursor. All characters to the right of the cursor are moved to the right, with the possibility of the rightmost characters on the line being lost. The cursor position does not change (after moving to <i>y</i>, <i>x</i>, if specified). (This does not imply use of the hardware insert character feature.) The four routines with <i>n</i> as the last argument insert at most <i>n</i> characters. If <i>n</i> <=0, then the entire string is inserted.</p> <p>If a character in <i>str</i> is a tab, newline, carriage return or backspace, the cursor is moved appropriately within the window. A newline also does a clrtoeol() before moving. Tabs are considered to be at every eighth column. If a character in <i>str</i> is another control character, it is drawn in the ^ X notation. Calling winch() after adding a control character (and moving to it, if necessary) does not return the control character, but instead returns the representation of the control character.</p> |

RETURN VALUES

All routines return the integer `ERR` upon failure and an integer value other than `ERR` upon successful completion.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`curs_clear(3X)`, `curs_inch(3X)`, `curses(3X)`, `attributes(5)`

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

Note that all but `winsnstr()` may be macros.

NAME curs_instr, instr, innstr, winstr, winnstr, mvinstr, mvinnstr, mvwinstr, mvwinnstr – get a string of characters from a curses window

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lcurses
[
library
... ]
#include <curses.h>

int instr(char * str);

int innstr(char * str, int n);

int winstr(WINDOW * win, char * str);

int winnstr(WINDOW * win, char * str, int n);

int mvinstr(int y, int x, char * str);

int mvinnstr(int y, int x, char * str, int n);

int mvwinstr(WINDOW * win, int y, int x, char * str);

int mvwinnstr(WINDOW * win, int y, int x, char * str, int n);
```

DESCRIPTION These routines return a string of characters in *str*, starting at the current cursor position in the named window and ending at the right margin of the window. Attributes are stripped from the characters. The four functions with *n* as the last argument return the string at most *n* characters long.

RETURN VALUES All routines return the integer `ERR` upon failure and an integer value other than `ERR` upon successful completion.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `curses(3X)`, `attributes(5)`

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

Note that all routines except **winnstr()** may be macros.

| NAME | curs_inswch, inswch, winswch, mvinswch, mvwinswch – insert a <code>wchar_t</code> character before the character under the cursor in a curses window | | | | |
|---------------------|---|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lcurses [library ..] #include <curses.h> int inswch(chtype wch); int winswch(WINDOW * win, chtype wch); int mvinswch(int y, int x, chtype wch); int mvwinswch(WINDOW * win, int y, int x, chtype wch);</pre> | | | | |
| DESCRIPTION | These routines insert the character <code>wch</code> , holding a <code>wchar_t</code> character, before the character under the cursor. All characters to the right of the cursor are moved one space to the right, with the possibility of the rightmost character on the line being lost. The cursor position does not change (after moving to <code>y</code> , <code>x</code> , if specified). (This does not imply use of the hardware insert character feature.) | | | | |
| RETURN VALUE | All routines return the integer <code>ERR</code> upon failure and an integer value other than <code>ERR</code> upon successful completion. | | | | |
| ATTRIBUTES | See <code>attributes(5)</code> for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | <code>curses(3X)</code> , <code>attributes(5)</code> | | | | |
| NOTES | <p>The header file <code><curses.h></code> automatically includes the header files <code><stdio.h></code>, <code><unctrl.h></code> and <code><widec.h></code>.</p> <p>Note that <code>inswch()</code>, <code>mvinswch()</code>, and <code>mvwinswch()</code> may be macros.</p> <p>None of these routines can use the color attribute in <code>chtype</code>.</p> | | | | |

| | |
|--------------------|---|
| NAME | curs_inswstr, inswstr, insnwstr, winswstr, winsnwstr, mvinswstr, mvinsnwstr, mvwinswstr, mvwinsnwstr – insert <code>wchar_t</code> string before character under the cursor in a curses window |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ..] #include <curses.h> int inswstr(wchar_t * wstr); int insnwstr(wchar_t * wstr, int n); int winswstr(WINDOW * win, wchar_t * wstr); int winsnwstr(WINDOW * win, wchar_t * wstr, int n); int mvinswstr(int y, int x, wchar_t * wstr); int mvinsnwstr(int y, int x, wchar_t * wstr, int n); int mvwinswstr(WINDOW * win, int y, int x, wchar_t * wstr); int mvwinsnwstr(WINDOW * win, int y, int x, wchar_t * wstr, int n);</pre> |
| DESCRIPTION | <p>These routines insert a <code>wchar_t</code> character string (as many <code>wchar_t</code> characters as will fit on the line) before the character under the cursor. All characters to the right of the cursor are moved to the right, with the possibility of the rightmost characters on the line being lost. The cursor position does not change (after moving to <code>y</code>, <code>x</code>, if specified). (This does not imply use of the hardware insert character feature.) The four routines with <code>n</code> as the last argument insert at most <code>n</code> <code>wchar_t</code> characters. If <code>n <= 0</code>, then the entire string is inserted.</p> <p>If a character in <code>wstr</code> is a tab, newline, carriage return, or backspace, the cursor is moved appropriately within the window. A newline also does a <code>clrtoeol(3X)</code> before moving. Tabs are considered to be at every eighth column. If a character in <code>wstr</code> is another control character, it is drawn in the <code>^X</code> notation. Calling <code>winwch(3X)</code> after adding a control character (and moving to it, if necessary) does not return the control character, but instead returns the representation of the control character.</p> |

RETURN VALUE All routines return the integer `ERR` upon failure and an integer value other than `ERR` upon successful completion.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `clrtoeol(3X)` , `curses(3X)` , `winch(3X)` , `attributes(5)`

NOTES The header file `<curses.h>` automatically includes the header files `<stdio.h>` , `<unctrl.h>` and `<widec.h>` .

Note that all but `winsnwstr()` may be macros.

NAME curs_inwch, inwch, winwch, mvwinch, mvwinwch – get a wchar_t character and its attributes from a curses window

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lcurses
[
library
.. ]
#include <curses.h>

ctype inwch(void);

ctype winwch(WINDOW * win);

ctype mvwinch(int y, int x);

ctype mvwinwch(WINDOW * win, int y, int x);
```

DESCRIPTION These routines return the wchar_t character, of type ctype , at the current position in the named window. If any attributes are set for that position, their values are OR-ed into the value returned. Constants defined in <curses.h> can be used with the logical AND (&)operator to extract the character or attributes alone.

Attributes The following bit-masks may be AND-ed with characters returned by winwch() .

```
A
_WCHARTEXT          Bit-mask to extract character
A_WATTRIBUTES      Bit-mask to extract attributes
```

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO curses(3X) , attributes(5)

NOTES | The header file `<curses.h>` automatically includes the header files `<stdio.h>` , `<unctrl.h>` and `<widec.h>` .

Note that all of these routines may be macros.

None of these routines can use the color attribute in `chtype` .

NAME curs_inwchstr, inwchstr, inwchnstr, winwchstr, winwchnstr, mvinwchstr, mvinwchnstr, mvwinwchstr, mvwinwchnstr – get a string of wchar_t characters (and attributes) from a curses window

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lcurses
[
library
.. ]
#include <curses.h>

int inwchstr(chtype * wchstr);
int inwchnstr(chtype * wchstr, int n);
int winwchstr(WINDOW * win, chtype * wchstr);
int winwchnstr(WINDOW * win, chtype * wchstr, int n);
int mvinwchstr(int y, int x, chtype * wchstr);
int mvinwchnstr(int y, int x, chtype * wchstr, int n);
int mvwinwchstr(WINDOW * win, int y, int x, chtype * wchstr);
int mvwinwchnstr(WINDOW * win, int y, int x, chtype * wchstr, int n);
```

DESCRIPTION These routines return a string of type chtype , holding wchar_t characters, starting at the current cursor position in the named window and ending at the right margin of the window. The four functions with n as the last argument, return the string at most n wchar_t characters long. Constants defined in <curses.h> can be used with the logical AND (&)operator to extract the wchar_t character or the attribute alone from any position in the wchstr (see curs_inwch(3X)).

RETURN VALUE All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.

ATTRIBUTES See attributes(5) for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO | `curses(3X)` , `curs_inwch(3X)` , `attributes(5)`

NOTES | The header file `<curses.h>` automatically includes the header files `<stdio.h>` , `<unctrl.h>` and `<widec.h>` .

Note that all routines except `winwchnstr()` may be macros.

None of these routines can use the color attribute in `chtype` .

NAME curs_inwstr, inwstr, innwstr, winwstr, winnwstr, mvinwstr, mvinnwstr, mvwinwstr, mvwinnwstr – get a string of wchar_t characters from a curses window

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lcurses
[
library
.. ]
#include <curses.h>

int inwstr(wchar_t * wstr);

int innwstr(wchar_t * wstr, int n);

int winwstr(WINDOW * win, wchar_t * wstr);

int winnwstr(WINDOW * win, wchar_t * wstr, int n);

int mvinwstr(int y, int x, wchar_t * wstr);

int mvinnwstr(int y, int x, wchar_t * wstr, int n);

int mvwinwstr(WINDOW * win, int y, int x, wchar_t * wstr);

int mvwinnwstr(WINDOW * win, int y, int x, wchar_t * wstr, int n);
```

DESCRIPTION These routines return the string of wchar_t characters in wstr starting at the current cursor position in the named window and ending at the right margin of the window. Attributes are stripped from the characters. The four functions with n as the last argument return the string at most n wchar_t characters long.

RETURN VALUES All routines return the integer ERR upon failure and an integer value other than ERR upon successful completion.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO curses(3X), attributes(5)

NOTES | The header file `<curses.h>` automatically includes the header files `<stdio.h>` , `<unctrl.h>` and `<widec.h>` .

Note that all routines except **winnwstr()** may be macros.

| | |
|--------------------|--|
| NAME | curs_kernel, def_prog_mode, def_shell_mode, reset_prog_mode, reset_shell_mode, resetty, savetty, getsyx, setsyx, ripoffline, curs_set, napms - low-level curses routines |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> int def_prog_mode(void); int def_shell_mode(void); int reset_prog_mode(void); int reset_shell_mode(void); int resetty(void); int savetty(void); int getsyx(int y, int x); int setsyx(int y, int x); int ripoffline(int <i>line</i>, int (* <i>init</i>)(WINDOW *, int)); int curs_set(int <i>visibility</i>); int napms(int <i>ms</i>);</pre> |
| DESCRIPTION | <p>The following routines give low-level access to various <code>curses</code> functionality. These routines typically are used inside library routines.</p> <p>The <code>def_prog_mode()</code> and <code>def_shell_mode()</code> routines save the current terminal modes as the “program” (in <code>curses</code>) or “shell” (not in <code>curses</code>) state for use by the <code>reset_prog_mode()</code> and <code>reset_shell_mode()</code> routines. This is done automatically by <code>initscr()</code>.</p> <p>The <code>reset_prog_mode()</code> and <code>reset_shell_mode()</code> routines restore the terminal to “program” (in <code>curses</code>) or “shell” (out of <code>curses</code>) state. These are done automatically by <code>endwin()</code> and, after an <code>endwin()</code>, by <code>doupdate()</code>, so they normally are not called.</p> |

The **resetty()** and **savetty()** routines save and restore the state of the terminal modes. **savetty()** saves the current state in a buffer and **resetty()** restores the state to what it was at the last call to **savetty()** .

With the **getsyx()** routine, the current coordinates of the virtual screen cursor are returned in *y* and *x*. If **leaveok()** is currently **TRUE** , then -1 , -1 is returned. If lines have been removed from the top of the screen, using **ripline()** , *y* and *x* include these lines; therefore, *y* and *x* should be used only as arguments for **setsyx()** .

With the **setsyx()** routine, the virtual screen cursor is set to *y* , *x* . If *y* and *x* are both -1 , then **leaveok()** is set. The two routines **getsyx()** and **setsyx()** are designed to be used by a library routine, which manipulates *curses* windows but does not want to change the current position of the program's cursor. The library routine would call **getsyx()** at the beginning, do its manipulation of its own windows, do a **wnoutrefresh()** on its windows, call **setsyx()** , and then call **doupdate()** .

The **ripline()** routine provides access to the same facility that **slk_init()** (see **curs_slk(3X)**) uses to reduce the size of the screen. **ripline()** must be called before **initscr()** or **newterm()** is called. If *line* is positive, a line is removed from the top of **stdscr()** ; if *line* is negative, a line is removed from the bottom. When this is done inside **initscr()** , the routine **init()** (supplied by the user) is called with two arguments: a window pointer to the one-line window that has been allocated and an integer with the number of columns in the window. Inside this initialization routine, the integer variables **LINES** and **COLS** (defined in `<curses.h>`) are not guaranteed to be accurate and **wrefresh()** or **doupdate()** must not be called. It is allowable to call **wnoutrefresh()** during the initialization routine.

ripline() can be called up to five times before calling **initscr()** or **newterm()** .

With the **curs_set()** routine, the cursor state is set to invisible, normal, or very visible for *visibility* equal to 0 , 1 , or 2 respectively. If the terminal supports the *visibility* requested, the previous *cursor* state is returned; otherwise, **ERR** is returned.

The **napms()** routine is used to sleep for *ms* milliseconds.

RETURN VALUES

Except for **curs_set()** , these routines always return **OK** . **curs_set()** returns the previous cursor state, or **ERR** if the requested *visibility* is not supported.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`curs_initscr(3X)`, `curs_outopts(3X)`, `curs_refresh(3X)`,
`curs_scr_dump(3X)`, `curs_slk(3X)`, `curses(3X)`, `attributes(5)`

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

Note that `getsyx()` is a macro, so an ampersand (`&`) is not necessary before the variables `y` and `x`.

NAME | curs_move, move, wmove – move curses window cursor

SYNOPSIS | cc
| [
| *flag*
| ...]
| *file*
| ...
| -lcurses
| [
| *library*
| ...]
| #include <curses.h>

| int move(int y, int x);

| int wmove(WINDOW * win, int y, int x);

DESCRIPTION | With these routines, the cursor associated with the window is moved to line *y* and column *x* . This routine does not move the physical cursor of the terminal until **refresh()** is called. The position specified is relative to the upper left-hand corner of the window, which is (0,0).

RETURN VALUES | These routines return the integer `ERR` upon failure and an integer value other than `ERR` upon successful completion.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO | **curs_refresh(3X)** , **curses(3X)** , **attributes(5)**

NOTES | The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>` .

| Note that **move()** may be a macro.

| | |
|--------------------|--|
| NAME | curs_outopts, clearok, idlok, idcok, immedok, leaveok, setscrreg, wsetscreg, scrollok, nl, nonl – curses terminal output option control routines |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> int clearok(WINDOW * win, bool bf); int idlok(WINDOW * win, bool bf); void idcok(WINDOW * win, bool bf); void immedok(WINDOW * win, bool bf); int leaveok(WINDOW * win, bool bf); int setscreg(int top, int bot); int wsetscreg(WINDOW * win, int top, int bot); int scrollok(WINDOW * win, bool bf); int nl(void); int nonl(void);</pre> |
| DESCRIPTION | <p>These routines set options that deal with output within <code>curses</code>. All options are initially <code>FALSE</code>, unless otherwise stated. It is not necessary to turn these options off before calling <code>endwin()</code>.</p> <p>With the <code>clearok()</code> routine, if enabled (<code>bf</code> is <code>TRUE</code>), the next call to <code>wrefresh()</code> with this window will clear the screen completely and redraw the entire screen from scratch. This is useful when the contents of the screen are uncertain, or in some cases for a more pleasing visual effect. If the <code>win</code> argument to <code>clearok()</code> is the global variable <code>curscr()</code>, the next call to <code>wrefresh()</code> with any window causes the screen to be cleared and repainted from scratch.</p> <p>With the <code>idlok()</code> routine, if enabled (<code>bf</code> is <code>TRUE</code>), <code>curses</code> considers using the hardware insert/delete line feature of terminals so equipped. If disabled (<code>bf</code> is <code>FALSE</code>), <code>curses</code> very seldom uses this feature. (The insert/delete character feature is always considered.) This option should be enabled only if the</p> |

application needs insert/delete line, for example, for a screen editor. It is disabled by default because insert/delete line tends to be visually annoying when used in applications where it isn't really needed. If insert/delete line cannot be used, `curses` redraws the changed portions of all lines.

With the `idcok()` routine, if enabled (`bf` is `TRUE`), `curses` considers using the hardware insert/delete character feature of terminals so equipped. This is enabled by default.

With the `immedok()` routine, if enabled (`bf` is `TRUE`), any change in the window image, such as the ones caused by `waddch()` , `wclrtoebot()` , `wscrl()` , etc., automatically cause a call to `wrefresh()` . However, it may degrade the performance considerably, due to repeated calls to `wrefresh()` . It is disabled by default. Normally, the hardware cursor is left at the location of the window cursor being refreshed. The `leaveok()` option allows the cursor to be left wherever the update happens to leave it. It is useful for applications where the cursor is not used, since it reduces the need for cursor motions. If possible, the cursor is made invisible when this option is enabled.

The `setscrreg()` and `wsetscrreg()` routines allow the application programmer to set a software scrolling region in a window. `top` and `bot` are the line numbers of the top and bottom margin of the scrolling region. (Line 0 is the top line of the window.) If this option and `scrollok()` are enabled, an attempt to move off the bottom margin line causes all lines in the scrolling region to scroll up one line. Only the text of the window is scrolled. (Note that this has nothing to do with the use of a physical scrolling region capability in the terminal, like that in the VT100. If `idlok()` is enabled and the terminal has either a scrolling region or insert/delete line capability, they will probably be used by the output routines.)

The `scrollok()` option controls what happens when the cursor of a window is moved off the edge of the window or scrolling region, either as a result of a newline action on the bottom line, or typing the last character of the last line. If disabled, (`bf` is `FALSE`), the cursor is left on the bottom line. If enabled, (`bf` is `TRUE`), `wrefresh()` is called on the window, and the physical terminal and window are scrolled up one line. (Note that in order to get the physical scrolling effect on the terminal, it is also necessary to call `idlok()` .)

The `nl()` and `nonl()` routines control whether newline is translated into carriage return and linefeed on output, and whether return is translated into newline on input. Initially, the translations do occur. By disabling these translations using `nonl()` , `curses` is able to make better use of the linefeed capability, resulting in faster cursor motion.

RETURN VALUES

`setscrreg()` and `wsetscrreg()` return `OK` upon success and `ERR` upon failure. All other routines that return an integer always return `OK` .

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`curl_addch(3X)`, `curl_clear(3X)`, `curl_initscr(3X)`,
`curl_refresh(3X)`, `curl_scroll(3X)`, `curses(3X)`, `attributes(5)`

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

Note that `clearok()`, `leaveok()`, `scrollok()`, `idcok()`, `nl()`, `nonl()`, and `setscrreg()` may be macros.

The `immedok()` routine is useful for windows that are used as terminal emulators.

NAME | curs_overlay, overlay, overwrite, copywin – overlap and manipulate overlapped curses windows

SYNOPSIS | cc
 [*flag* ...]
file
 ...
 -lcurses
 [*library* ...]
 #include <curses.h>

```
int overlay(WINDOW * srcwin, WINDOW * dstwin);

int overwrite(WINDOW * srcwin, WINDOW * dstwin);

int copywin(WINDOW * srcwin, WINDOW * dstwin, int sminrow, int smincol, int
dminrow, int dmincol, int dmaxrow, int dmaxcol, int overlay);
```

DESCRIPTION | The **overlay()** and **overwrite()** routines overlay *srcwin* on top of *dstwin*. *srcwin* and *dstwin* are not required to be the same size; only text where the two windows overlap is copied. The difference is that **overlay()** is non-destructive (blanks are not copied) whereas **overwrite()** is destructive.

The **copywin()** routine provides a finer granularity of control over the **overlay()** and **overwrite()** routines. Like in the **prefresh()** routine, a rectangle is specified in the destination window, (*dminrow* , *dmincol*) and (*dmaxrow* , *dmaxcol*), and the upper-left-corner coordinates of the source window, (*sminrow* , *smincol*). If the argument *overlay* is `true` , then copying is non-destructive, as in **overlay()** .

RETURN VALUES | Routines that return an integer return `ERR` upon failure and an integer value other than `ERR` upon successful completion.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO | **curs_pad(3X)** , **curs_refresh(3X)** , **curses(3X)** , **attributes(5)**

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

Note that **overlay()** and `overwrite` may be macros.

| | |
|--------------------|--|
| NAME | curs_pad, newpad, subpad, prefresh, pnoutrefresh, pechochar, pechowchar – create and display curses pads |
| SYNOPSIS | <pre>cc [flag ...] file ... -lcurses [library ..] #include <curses.h> WINDOW * newpad(int nlines, int ncols); WINDOW * subpad(WINDOW * orig, int nlines, int ncols, int begin_y, int begin_x); int prefresh(WINDOW * pad, int pminrow, int pmincol, int sminrow, int smincol, int smaxrow, int smaxcol); int pnoutrefresh(WINDOW * pad, int pminrow, int pmincol, int sminrow, int smincol, int smaxrow, int smaxcol); int pechochar(WINDOW * pad, chtype ch); int pechowchar(WINDOW * pad, chtype wch);</pre> |
| DESCRIPTION | <p>The newpad() routine creates and returns a pointer to a new pad data structure with the given number of lines, <i>nlines</i>, and columns, <i>ncols</i>. A pad is like a window, except that it is not restricted by the screen size, and is not necessarily associated with a particular part of the screen. Pads can be used when a large window is needed, and only a part of the window will be on the screen at one time. Automatic refreshes of pads (for example, from scrolling or echoing of input) do not occur. It is not legal to call wrefresh(3X) with a <i>pad</i> as an argument; the routines prefresh() or pnoutrefresh() should be called instead. Note that these routines require additional parameters to specify the part of the pad to be displayed and the location on the screen to be used for the display.</p> <p>The subpad() routine creates and returns a pointer to a subwindow within a pad with the given number of lines, <i>nlines</i>, and columns, <i>ncols</i>. Unlike subwin(3X), which uses screen coordinates, the window is at position (<i>begin_x</i>, <i>begin_y</i>) on the pad. The window is made in the middle of the window <i>orig</i>, so that changes made to one window affect both windows. During the use of this routine, it will often be necessary to call touchwin(3X) or touchline(3X) on <i>orig</i> before calling prefresh().</p> |

The **prefresh()** and **pnoutrefresh()** routines are analogous to **wrefresh(3X)** and **wnoutrefresh(3X)** except that they relate to pads instead of windows. The additional parameters are needed to indicate what part of the pad and screen are involved. *pminrow* and *pmincol* specify the upper left-hand corner of the rectangle to be displayed in the pad. *sminrow*, *smincol*, *smaxrow*, and *smaxcol* specify the edges of the rectangle to be displayed on the screen. The lower right-hand corner of the rectangle to be displayed in the pad is calculated from the screen coordinates, since the rectangles must be the same size. Both rectangles must be entirely contained within their respective structures. Negative values of *pminrow*, *pmincol*, *sminrow*, or *smincol* are treated as if they were zero.

The **pechochar()** routine is functionally equivalent to a call to **addch(3X)** followed by a call to **refresh(3X)**, a call to **waddch(3X)** followed by a call to **wrefresh(3X)**, or a call to **waddch(3X)** followed by a call to **prefresh()**. The knowledge that only a single character is being output is taken into consideration and, for non-control characters, a considerable performance gain might be seen by using these routines instead of their equivalents. In the case of **pechochar()**, the last location of the pad on the screen is reused for the arguments to **prefresh()**.

RETURN VALUES

Routines that return an integer return **ERR** upon failure and an integer value other than **ERR** upon successful completion.

Routines that return pointers return **NULL** on error.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

addch(3X), **curses(3X)**, **refresh(3X)**, **subwin(3X)**, **touchline(3X)**, **touchwin(3X)**, **waddch(3X)**, **wnoutrefresh(3X)**, **wrefresh(3X)**, **attributes(5)**

NOTES

The header file `<curses.h>` automatically includes the header files `<stdio.h>`, `<unctrl.h>` and `<widec.h>`.

Note that **pechochar()** may be a macro.

| NAME | curs_printw, printw, wprintw, mvprintw, mvwprintw, vwprintw – print formatted output in curses windows | | | | |
|----------------------|---|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre> cc [flag ...] file ... -lcurses [library ...] #include <curses.h> int printw(char * fmt, /* arg */ ...); int wprintw(WINDOW * win, char * fmt, /* arg */ ...); int mvprintw(int y, int x, char * fmt, /* arg */ ...); int mvwprintw(WINDOW * win, int y, int x, char * fmt, /* arg */ ...); #include <varargs.h> int vwprintw(WINDOW * win, char * fmt, /* varlist */ ...); </pre> | | | | |
| DESCRIPTION | <p>The printw(), wprintw(), mvprintw(), and mvwprintw() routines are analogous to printf() (see printf(3S)). In effect, the string that would be output by printf() is output instead as though waddstr() were used on the given window.</p> <p>The vwprintw() routine is analogous to vprintf() (see vprintf(3S)) and performs a wprintw() using a variable argument list. The third argument is a <code>va_list</code>, a pointer to a list of arguments, as defined in <code><varargs.h></code>.</p> | | | | |
| RETURN VALUES | All routines return the integer <code>ERR</code> upon failure and an integer value other than <code>ERR</code> upon successful completion. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | curses(3X) , printf(3S) , vprintf(3S) , attributes(5) | | | | |

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

| | |
|--------------------|--|
| NAME | curs_refresh, refresh, wrefresh, wnoutrefresh, doupdate, redrawwin, wredrawln – refresh curses windows and lines |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> int refresh(void); int wrefresh(WINDOW * win); int wnoutrefresh(WINDOW * win); int doupdate(void); int redrawwin(WINDOW * win); int wredrawln(WINDOW * win, int beg_line, int num_lines);</pre> |
| DESCRIPTION | <p>The refresh() and wrefresh() routines (or wnoutrefresh() and doupdate()) must be called to get any output on the terminal, as other routines merely manipulate data structures. The routine wrefresh() copies the named window to the physical terminal screen, taking into account what is already there in order to do optimizations. The refresh() routine is the same, using <code>stdscr</code> as the default window. Unless leaveok() has been enabled, the physical cursor of the terminal is left at the location of the cursor for that window.</p> <p>The wnoutrefresh() and doupdate() routines allow multiple updates with more efficiency than wrefresh() alone. In addition to all the window structures, <code>curses</code> keeps two data structures representing the terminal screen: a physical screen, describing what is actually on the screen, and a virtual screen, describing what the programmer wants to have on the screen.</p> <p>The routine wrefresh() works by first calling wnoutrefresh(), which copies the named window to the virtual screen, and then calling doupdate(), which compares the virtual screen to the physical screen and does the actual update. If the programmer wishes to output several windows at once, a series of calls to wrefresh() results in alternating calls to wnoutrefresh() and doupdate(), causing several bursts of output to the screen. By first calling wnoutrefresh() for each window, it is then possible to call doupdate() once, resulting in only</p> |

one burst of output, with fewer total characters transmitted and less CPU time used. If the *win* argument to **wrefresh()** is the global variable *curscr*, the screen is immediately cleared and repainted from scratch.

The **redrawwin()** routine indicates to *curses* that some screen lines are corrupted and should be thrown away before anything is written over them. These routines could be used for programs such as editors, which want a command to redraw some part of the screen or the entire screen. The routine **redrawln()** is preferred over **redrawwin()** where a noisy communication line exists and redrawing the entire window could be subject to even more communication noise. Just redrawing several lines offers the possibility that they would show up unblemished.

RETURN VALUES

All routines return the integer `ERR` upon failure and an integer value other than `ERR` upon successful completion.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

curs_outopts(3X), **curses(3X)**, **attributes(5)**

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

Note that **refresh()** and **redrawwin()** may be macros.

| NAME | curs_scanw, scanw, wscanw, mvscanw, mvwscanw, vwscanw – convert formatted input from a curses widow | | | | |
|----------------------|---|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lcurses [library ...] #include <curses.h> int scanw(char * fmt, /* arg */ ...); int wscanw(WINDOW * win, char * fmt, /* arg */ ...); int mvscanw(int y, int x, char * fmt, /* arg */ ...); int mvwscanw(WINDOW * win, int y, int x, char * fmt, /* arg */ ...); int vwscanw(WINDOW * win, char * fmt, va_list varlist);</pre> | | | | |
| DESCRIPTION | <p>The scanw(), wscanw(), and mvscanw() routines correspond to scanf() (see scanf(3S)). The effect of these routines is as though wgetstr() were called on the window, and the resulting line used as input for the scan. Fields which do not map to a variable in the <code>fmt</code> field are lost.</p> <p>The vwscanw() routine is similar to vwprintw() in that it performs a wscanw() using a variable argument list. The third argument is a <code>va_list</code>, a pointer to a list of arguments, as defined in <code><varargs.h></code>.</p> | | | | |
| RETURN VALUES | <p>vwscanw() returns <code>ERR</code> on failure and an integer equal to the number of fields scanned on success.</p> <p>Applications may interrogate the return value from the scanw(), wscanw(), mvscanw(), and mvwscanw() routines to determine the number of fields which were mapped in the call.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |

SEE ALSO | `curs_getstr(3X)`, `curs_printw(3X)`, `curses(3X)`, `scanf(3S)`,
| `attributes(5)`

NOTES | The header `<curses.h>` automatically includes the headers `<stdio.h>` and
| `<unctrl.h>`.

| | |
|----------------------|--|
| NAME | curs_scr_dump, scr_dump, scr_restore, scr_init, scr_set – read (write) a curses screen from (to) a file |
| SYNOPSIS | <pre>cc [flag ...] file ... -lcurses [library ...] #include <curses.h> int scr_dump(char * filename); int scr_restore(char * filename); int scr_init(char * filename); int scr_set(char * filename);</pre> |
| DESCRIPTION | <p>With the scr_dump() routine, the current contents of the virtual screen are written to the file <i>filename</i> .</p> <p>With the scr_restore() routine, the virtual screen is set to the contents of <i>filename</i> , which must have been written using scr_dump() . The next call to doupdate() restores the screen to the way it looked in the dump file.</p> <p>With the scr_init() routine, the contents of <i>filename</i> are read in and used to initialize the <i>curses</i> data structures about what the terminal currently has on its screen. If the data is determined to be valid, <i>curses</i> bases its next update of the screen on this information rather than clearing the screen and starting from scratch. scr_init() is used after initscr() or a system(3S) call to share the screen with another process which has done a scr_dump() after its endwin() call. The data is declared invalid if the time-stamp of the tty is old or the terminfo capabilities rmcup() and nrrmc() exist.</p> <p>The scr_set() routine is a combination of scr_restore() and scr_init() . It tells the program that the information in <i>filename</i> is what is currently on the screen, and also what the program wants on the screen. This can be thought of as a screen inheritance function.</p> <p>To read (write) a window from (to) a file, use the getwin() and putwin() routines (see curs_util(3X)).</p> |
| RETURN VALUES | All routines return the integer ERR upon failure and OK upon success. |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

curlscr_init(3X), **curlscr_refresh(3X)**, **curlscr_util(3X)**, **curses(3X)**, **system(3S)**, **attributes(5)**

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

Note that **scr_init()**, **scr_set()**, and **scr_restore()** may be macros.

| NAME | curs_scroll, scroll, scl, wscr1 – scroll a curses window | | | | |
|----------------------|---|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lcurses [library ...] #include <curses.h> int scroll(WINDOW * win); int scl(int n); int wscr1(WINDOW * win, int n);</pre> | | | | |
| DESCRIPTION | <p>With the scroll() routine, the window is scrolled up one line. This involves moving the lines in the window data structure. As an optimization, if the scrolling region of the window is the entire screen, the physical screen is scrolled at the same time.</p> <p>With the scl() and wscr1() routines, for positive <i>n</i> scroll the window up <i>n</i> lines (line <i>i+n</i> becomes <i>i</i>); otherwise scroll the window down <i>n</i> lines. This involves moving the lines in the window character image structure. The current cursor position is not changed.</p> <p>For these functions to work, scrolling must be enabled via scrollok().</p> | | | | |
| RETURN VALUES | All routines return the integer <code>ERR</code> upon failure and an integer value other than <code>ERR</code> upon successful completion. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | curs_outopts(3X) , curses(3X) , attributes(5) | | | | |
| NOTES | <p>The header <code><curses.h></code> automatically includes the headers <code><stdio.h></code> and <code><unctrl.h></code>.</p> <p>Note that scl() and scroll() may be macros.</p> | | | | |

| | |
|----------------------|--|
| NAME | curs_set – set visibility of cursor |
| SYNOPSIS | #include <curses.h> int curs_set (int <i>visibility</i>); |
| PARAMETERS | <i>visibility</i> Is a value of 0 (invisible), 1 (normal), or 2 (very visible). |
| DESCRIPTION | The curs_set() function sets the visibility of the cursor to invisible (0), normal (1), or very visible (2). The exact appearance of normal and very visible cursors is terminal dependent. |
| RETURN VALUES | If the terminal supports the mode specified by the <i>visibility</i> parameter, the curs_set() function returns the previous cursor state. Otherwise, it returns <code>ERR</code> . |
| ERRORS | None. |

| | |
|--------------------|--|
| NAME | curs_slk, slk_init, slk_set, slk_refresh, slk_noutrefresh, slk_label, slk_clear, slk_restore, slk_touch, slk_attron, slk_attrset, slk_attroff – curses soft label routines |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> int slk_init(int <i>fmt</i>); int slk_set(int <i>labnum</i>, char * <i>label</i>, int <i>fmt</i>); int slk_refresh(void); int slk_noutrefresh(void); char * slk_label(int <i>labnum</i>); int slk_clear(void); int slk_restore(void); int slk_touch(void); int slk_attron(chtype <i>attrs</i>); int slk_attrset(chtype <i>attrs</i>); int slk_attroff(chtype <i>attrs</i>);</pre> |
| DESCRIPTION | <p>curses manipulates the set of soft function-key labels that exist on many terminals. For those terminals that do not have soft labels, curses takes over the bottom line of <code>stdscr</code>, reducing the size of <code>stdscr</code> and the variable <code>LINES</code>. curses standardizes on eight labels of up to eight characters each.</p> <p>To use soft labels, the <code>slk_init()</code> routine must be called before <code>initscr()</code> or <code>newterm()</code> is called. If <code>initscr()</code> eventually uses a line from <code>stdscr</code> to emulate the soft labels, then <code>fmt</code> determines how the labels are arranged on the screen. Setting <code>fmt</code> to 0 indicates a 3-2-3 arrangement of the labels; 1 indicates a 4-4 arrangement.</p> <p>With the <code>slk_set()</code> routine, <code>labnum</code> is the label number, from 1 to 8. <code>label</code> is the string to be put on the label, up to eight characters in length. A null string or a</p> |

null pointer sets up a blank label. *fmt* is either 0, 1, or 2, indicating whether the label is to be left-justified, centered, or right-justified, respectively, within the label.

The **slk_refresh()** and **slk_noutrefresh()** routines correspond to the **wrefresh()** and **wnoutrefresh()** routines.

With the **slk_label()** routine, the current label for label number *labnum* is returned with leading and trailing blanks stripped.

With the **slk_clear()** routine, the soft labels are cleared from the screen.

With the **slk_restore()** routine, the soft labels are restored to the screen after a **slk_clear()** is performed.

With the **slk_touch()** routine, all the soft labels are forced to be output the next time a **slk_noutrefresh()** is performed.

The **slk_attron()**, **slk_attrset()**, and **slk_attroff()** routines correspond to **attron()**, **attrset()**, and **attroff()**. They have an effect only if soft labels are simulated on the bottom line of the screen.

RETURN VALUES

Routines that return an integer return **ERR** upon failure and an integer value other than **ERR** upon successful completion.

slk_label() returns **NULL** on error.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

curs_attr(3X), **curs_initscr(3X)**, **curs_refresh(3X)**, **curses(3X)**, **attributes(5)**

NOTES

The header **<curses.h>** automatically includes the headers **<stdio.h>** and **<unctrl.h>**.

Most applications would use **slk_noutrefresh()** because a **wrefresh()** is likely to follow soon.

| | |
|--------------------|---|
| NAME | curs_termattrs, baudrate, erasechar, has_ic, has_il, killchar, longname, termattrs, termname – curses environment query routines |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> int baudrate(void); char erasechar(void); int has_ic(void); int has_il(void); char killchar(void); char * longname(void); cctype termattrs(void); char * termname(void);</pre> |
| DESCRIPTION | <p>The baudrate() routine returns the output speed of the terminal. The number returned is in bits per second, for example 9600 , and is an integer.</p> <p>With the erasechar() routine, the user's current erase character is returned.</p> <p>The has_ic() routine is true if the terminal has insert- and delete-character capabilities.</p> <p>The has_il() routine is true if the terminal has insert- and delete-line capabilities, or can simulate them using scrolling regions. This might be used to determine if it would be appropriate to turn on physical scrolling using scrollok() .</p> <p>With the killchar() routine, the user's current line kill character is returned.</p> <p>The longname() routine returns a pointer to a static area containing a verbose description of the current terminal. The maximum length of a verbose description is 128 characters. It is defined only after the call to initscr() or newterm() . The area is overwritten by each call to newterm() and is not</p> |

restored by **set_term()** , so the value should be saved between calls to **newterm()** if **longname()** is going to be used with multiple terminals.

If a given terminal doesn't support a video attribute that an application program is trying to use, *curses* may substitute a different video attribute for it. The **termattrs()** function returns a logical OR of all video attributes supported by the terminal. This information is useful when a *curses* program needs complete control over the appearance of the screen.

The **termname()** routine returns the value of the environment variable TERM (truncated to 14 characters).

RETURN VALUES

longname() and **termname()** return NULL on error.

Routines that return an integer return ERR upon failure and an integer value other than ERR upon successful completion.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

curs_initscr(3X) , **curs_outopts(3X)** , **curses(3X)** , **attributes(5)**

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>` .

Note that **termattrs()** may be a macro.

| | |
|----------------------|--|
| NAME | curs_termcap, tgetent, tgetflag, tgetnum, tgetstr, tgoto, tputs – curses interfaces (emulated) to the termcap library |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> #include <term.h> int tgetent(char * <i>bp</i>, char * <i>name</i>); int tgetflag(char <i>id</i> [2]); int tgetnum(char <i>id</i> [2]); char * tgetstr(char <i>id</i> [2], char ** <i>area</i>); char * tgoto(char * <i>cap</i>, int <i>col</i>, int <i>row</i>); int tputs(char * <i>str</i>, int <i>affcnt</i>, int (* <i>putc</i>)(void));</pre> |
| DESCRIPTION | <p>These routines are included as a conversion aid for programs that use the <i>termcap</i> library. Their parameters are the same and the routines are emulated using the <i>terminfo</i> database. These routines are supported at Level 2 and should not be used in new applications.</p> <p>The tgetent() routine looks up the termcap entry for <i>name</i> . The emulation ignores the buffer pointer <i>bp</i> .</p> <p>The tgetflag() routine gets the boolean entry for <i>id</i> .</p> <p>The tgetnum() routine gets the numeric entry for <i>id</i> .</p> <p>The tgetstr() routine returns the string entry for <i>id</i> . Use tputs() to output the returned string.</p> <p>The tgoto() routine instantiates the parameters into the given capability. The output from this routine is to be passed to tputs() .</p> <p>The tputs() routine is described on the curs_terminfo(3X) manual page.</p> |
| RETURN VALUES | Routines that return an integer return ERR upon failure and an integer value other than ERR upon successful completion. |

Routines that return pointers return `NULL` on error.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`curl_terminfo(3X)`, `curses(3X)`, `putc(3S)`, `attributes(5)`

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

| | |
|--------------------|--|
| NAME | curs_terminfo, setupterm, setterm, set_curterm, del_curterm, restartterm, tparm, tputs, putp, vidputs, vidattr, mvcur, tigetflag, tigetnum, tigetstr – curses interfaces to terminfo database |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> #include <term.h> int setupterm(char * term, int <i>filde</i>s, int * <i>erret</i>); int setterm(char * term); int set_curterm(TERMINAL * <i>nterm</i>); int del_curterm(TERMINAL * <i>oterm</i>); int restartterm(char * term, int <i>filde</i>s, int * <i>erret</i>); char * tparm(char * <i>str</i>, long int <i>p1</i>, long int <i>p2</i>, long int <i>p3</i>, long int <i>p4</i>, long int <i>p5</i>, long int <i>p6</i>, long int <i>p7</i>, long int <i>p8</i>, long int <i>p9</i>); int tputs(char * <i>str</i>, int <i>affcnt</i>, int (* <i>putc</i>)(<i>char</i>)); int putp(char * <i>str</i>); int vidputs(chtype <i>attrs</i>, int (* <i>putc</i>)(<i>char</i>)); int vidattr(chtype <i>attrs</i>); int mvcur(int <i>oldrow</i>, int <i>oldcol</i>, int <i>newrow</i>, int <i>newcol</i>); int tigetflag(char * <i>capname</i>); int tigetnum(char * <i>capname</i>); char * tigetstr(char * <i>capname</i>);</pre> |
| DESCRIPTION | <p>These low-level routines must be called by programs that have to deal directly with the <i>terminfo</i> database to handle certain terminal capabilities, such as programming function keys. For all other functionality, <i>curses</i> routines are more suitable and their use is recommended.</p> |

Initially, **setupterm()** should be called. Note that **setupterm()** is automatically called by **initscr()** and **newterm()**. This defines the set of terminal-dependent variables (listed in **terminfo(4)**). The *terminfo* variables `lines` and `columns` are initialized by **setupterm()** as follows: If `use_env(FALSE)` has been called, values for `lines` and `columns` specified in *terminfo* are used. Otherwise, if the environment variables `LINES` and `COLUMNS` exist, their values are used. If these environment variables do not exist and the program is running in a window, the current window size is used. Otherwise, if the environment variables do not exist, the values for `lines` and `columns` specified in the *terminfo* database are used.

The headers `<curses.h>` and `<term.h>` should be included (in this order) to get the definitions for these strings, numbers, and flags. Parameterized strings should be passed through **tparm()** to instantiate them. All *terminfo* strings (including the output of **tparm()**) should be printed with **tputs()** or **putp()**. Call the **reset_shell_mode()** routine to restore the tty modes before exiting (see **curs_kernel(3X)**). Programs which use cursor addressing should output `enter_ca_mode` upon startup and should output `exit_ca_mode` before exiting. Programs desiring shell escapes should call **reset_shell_mode** and output `exit_ca_mode` before the shell is called and should output `enter_ca_mode` and call **reset_prog_mode** after returning from the shell.

The **setupterm()** routine reads in the *terminfo* database, initializing the *terminfo* structures, but does not set up the output virtualization structures used by *curses*. The terminal type is the character string *term*; if *term* is null, the environment variable `TERM` is used. All output is to file descriptor *fdes* which is initialized for output. If *errret* is not null, then **setupterm()** returns `OK` or `ERR` and stores a status value in the integer pointed to by *errret*. A status of `1` in *errret* is normal, `0` means that the terminal could not be found, and `-1` means that the *terminfo* database could not be found. If *errret* is null, **setupterm()** prints an error message upon finding an error and exits. Thus, the simplest call is:

```
setupterm((char *)0, 1, (int *)0);
```

which uses all the defaults and sends the output to `stdout`.

The **setterm()** routine is being replaced by **setupterm()**. The call:

```
setupterm( term , 1, (int *)0)
```

provides the same functionality as `setterm(term)`. The **setterm()** routine is included here for compatibility and is supported at Level 2.

The **set_curterm()** routine sets the variable `cur_term` to `nterm`, and makes all of the *terminfo* boolean, numeric, and string variables use the values from `nterm`.

The **del_curterm()** routine frees the space pointed to by `oterm` and makes it available for further use. If `oterm` is the same as `cur_term`, references to any of the *terminfo* boolean, numeric, and string variables thereafter may refer to invalid memory locations until another **setupterm()** has been called.

The **restartterm()** routine is similar to **setupterm()** and **initscr()**, except that it is called after restoring memory to a previous state. It assumes that the windows and the input and output options are the same as when memory was saved, but the terminal type and baud rate may be different.

The **tparm()** routine instantiates the string `str` with parameters `pi`. A pointer is returned to the result of `str` with the parameters applied.

The **tputs()** routine applies padding information to the string `str` and outputs it. The `str` must be a *terminfo* string variable or the return value from **tparm()**, **tgetstr()**, or **tgoto()**. `affcnt` is the number of lines affected, or 1 if not applicable. `putc` is a **putchar()**-like routine to which the characters are passed, one at a time.

The **putp()** routine calls `tputs(str, 1, putchar)`. Note that the output of **putpA()** always goes to `stdout`, not to the *fildev* specified in **setupterm()**.

The **vidputs()** routine displays the string on the terminal in the video attribute mode `attrs`, which is any combination of the attributes listed in **courses(3X)**. The characters are passed to the **putchar()**-like routine **putc()**.

The **vidattr()** routine is like the **vidputs()** routine, except that it outputs through **putchar()**.

The **mvcur()** routine provides low-level cursor motion.

The **tigetflag()**, **tigetnum()** and **tigetstr()** routines return the value of the capability corresponding to the *terminfo* `capname` passed to them, such as `xenl`.

With the **tigetflag()** routine, the value `-1` is returned if `capname` is not a boolean capability.

With the **tigetnum()** routine, the value `-2` is returned if `capname` is not a numeric capability.

With the **tigetstr()** routine, the value `(char *)-1` is returned if `capname` is not a string capability.

The `capname` for each capability is given in the table column entitled `capname` code in the capabilities section of **terminfo(4)**.

```
char *boolnames, *boolcodes, *boolfnames
char *numnames, *numcodes, *numfnames
char *strnames, *strcodes, *strfnames
```

These null-terminated arrays contain the *capnames*, the *termcap* codes, and the full C names, for each of the *terminfo* variables.

RETURN VALUES

All routines return the integer `ERR` upon failure and an integer value other than `ERR` upon successful completion, unless otherwise noted in the preceding routine descriptions.

Routines that return pointers always return `NULL` on error.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`curl_initscr(3X)`, `curl_kernel(3X)`, `curl_termcap(3X)`, `curses(3X)`, `putc(3S)`, `terminfo(4)`, `attributes(5)`

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

The `setupterm()` routine should be used in place of `setterm()`.

Note that `vidattr()` and `vidputs()` may be macros.

| | |
|--------------------|---|
| NAME | curs_touch, touchwin, touchline, untouchwin, wtouchln, is_linetouched, is_wintouched – curses refresh control routines |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> int touchwin(WINDOW * win); int touchline(WINDOW * win, int start, int count); int untouchwin(WINDOW * win); int wtouchln(WINDOW * win, int y, int n, int changed); int is_linetouched(WINDOW * win, int line); int is_wintouched(WINDOW * win);</pre> |
| DESCRIPTION | <p>The touchwin() and touchline() routines throw away all optimization information about which parts of the window have been touched, by pretending that the entire window has been drawn on. This is sometimes necessary when using overlapping windows, since a change to one window affects the other window, but the records of which lines have been changed in the other window do not reflect the change. The routine touchline() only pretends that <i>count</i> lines have been changed, beginning with line <i>start</i> .</p> <p>The untouchwin() routine marks all lines in the window as unchanged since the last call to wrefresh() .</p> <p>The wtouchln() routine makes <i>n</i> lines in the window, starting at line <i>y</i> , look as if they have (<i>changed</i> =1) or have not (<i>changed</i> =0) been changed since the last call to wrefresh() .</p> <p>The is_linetouched() and is_wintouched() routines return TRUE if the specified line/window was modified since the last call to wrefresh() ; otherwise they return FALSE . In addition, is_linetouched() returns ERR if <i>line</i> is not valid for the given window.</p> |

RETURN VALUES

All routines return the integer `ERR` upon failure and an integer value other than `ERR` upon successful completion, unless otherwise noted in the preceding routine descriptions.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`curl_refresh(3X)` , `curses(3X)` , `attributes(5)`

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>` .

Note that all routines except `wtouchln()` may be macros.

| | |
|--------------------|---|
| NAME | curs_util, unctrl, keyname, filter, use_env, putwin, getwin, delay_output, flushinp – curses miscellaneous utility routines |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> char * unctrl(chtype c); char * keyname(int c); int filter(void); void use_env(char <i>bool</i>); int putwin(WINDOW * <i>win</i>, FILE * <i>filep</i>); WINDOW * getwin(FILE * <i>filep</i>); int delay_output(int <i>ms</i>); int flushinp(void);</pre> |
| DESCRIPTION | <p>The unctrl() macro expands to a character string which is a printable representation of the character <i>c</i>. Control characters are displayed in the ^ X notation. Printing characters are displayed as is.</p> <p>With the keyname() routine, a character string corresponding to the key <i>c</i> is returned.</p> <p>The filter() routine, if used, is called before initscr() or newterm() are called. It makes <i>curses</i> think that there is a one-line screen. <i>curses</i> does not use any terminal capabilities that assume that they know on what line of the screen the cursor is positioned.</p> <p>The use_env() routine, if used, is called before initscr() or newterm() are called. When called with FALSE as an argument, the values of <i>lines</i> and <i>columns</i> specified in the <i>terminfo</i> database will be used, even if environment variables <i>LINES</i> and <i>COLUMNS</i> (used by default) are set, or if <i>curses</i> is running in a window (in which case default behavior would be to use the window size if <i>LINES</i> and <i>COLUMNS</i> are not set).</p> |

With the **putwin()** routine, all data associated with window *win* is written into the file to which *filep* points. This information can be later retrieved using the **getwin()** function.

The **getwin()** routine reads window related data stored in the file by **putwin()**. The routine then creates and initializes a new window using that data. It returns a pointer to the new window.

The **delay_output()** routine inserts an *ms* millisecond pause in output. This routine should not be used extensively because padding characters are used rather than a CPU pause.

The **flushinp()** routine throws away any typeahead that has been typed by the user and has not yet been read by the program.

RETURN VALUES

Except for **flushinp()**, routines that return an integer return `ERR` upon failure and an integer value other than `ERR` upon successful completion.

flushinp() always returns `OK`.

Routines that return pointers return `NULL` on error.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

curs_initscr(3X), **curs_scr_dump(3X)**, **curses(3X)**, **attributes(5)**

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

Note that **unctrl()** is a macro, which is defined in `<unctrl.h>`.

| | |
|--------------------|---|
| NAME | curs_window, newwin, delwin, mvwin, subwin, derwin, mvderwin, dupwin, wsyncup, syncok, wcursyncup, wsyncdown – create curses windows |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lcurses [<i>library</i> ...] #include <curses.h> WINDOW * newwin(int <i>nlines</i>, int <i>ncols</i>, int <i>begin_y</i>, int <i>begin_x</i>); int delwin(WINDOW * <i>win</i>); int mvwin(WINDOW * <i>win</i>, int <i>y</i>, int <i>x</i>); WINDOW * subwin(WINDOW * <i>orig</i>, int <i>nlines</i>, int <i>ncols</i>, int <i>begin_y</i>, int <i>begin_x</i>); WINDOW * derwin(WINDOW * <i>orig</i>, int <i>nlines</i>, int <i>ncols</i>, int <i>begin_y</i>, int <i>begin_x</i>); int mvderwin(WINDOW * <i>win</i>, int <i>par_y</i>, int <i>par_x</i>); WINDOW * dupwin(WINDOW * <i>win</i>); void wsyncup(WINDOW * <i>win</i>); int syncok(WINDOW * <i>win</i>, bool <i>bf</i>); void wcursyncup(WINDOW * <i>win</i>); void wsyncdown(WINDOW * <i>win</i>);</pre> |
| DESCRIPTION | <p>The newwin() routine creates and returns a pointer to a new window with the given number of lines, <i>nlines</i>, and columns, <i>ncols</i>. The upper left-hand corner of the window is at line <i>begin_y</i>, column <i>begin_x</i>. If either <i>nlines</i> or <i>ncols</i> is zero, they default to LINES — <i>begin_y</i> and COLS — <i>begin_x</i>. A new full-screen window is created by calling <code>newwin(0, 0, 0, 0)</code>.</p> <p>The delwin() routine deletes the named window, freeing all memory associated with it. Subwindows must be deleted before the main window can be deleted.</p> <p>The mvwin() routine moves the window so that the upper left-hand corner is at position (<i>x</i>, <i>y</i>). If the move would cause the window to be off the screen, it is an error and the window is not moved. Moving subwindows is allowed, but should be avoided.</p> |

The **subwin()** routine creates and returns a pointer to a new window with the given number of lines, *nlines*, and columns, *ncols*. The window is at position (*begin_y*, *begin_x*) on the screen. (This position is relative to the screen, and not to the window *orig*.) The window is made in the middle of the window *orig*, so that changes made to one window will affect both windows. The subwindow shares memory with the window *orig*. When using this routine, it is necessary to call **touchwin()** or **touchline()** on *orig* before calling **wrefresh()** on the subwindow.

The **derwin()** routine is the same as **subwin()**, except that *begin_y* and *begin_x* are relative to the origin of the window *orig* rather than the screen. There is no difference between the subwindows and the derived windows.

The **mvderwin()** routine moves a derived window (or subwindow) inside its parent window. The screen-relative parameters of the window are not changed. This routine is used to display different parts of the parent window at the same physical position on the screen.

The **dupwin()** routine creates an exact duplicate of the window *win*.

Each *curses* window maintains two data structures: the character image structure and the status structure. The character image structure is shared among all windows in the window hierarchy (that is, the window with all subwindows). The status structure, which contains information about individual line changes in the window, is private to each window. The routine **wrefresh()** uses the status data structure when performing screen updating. Since status structures are not shared, changes made to one window in the hierarchy may not be properly reflected on the screen.

The routine **wsyncup()** causes the changes in the status structure of a window to be reflected in the status structures of its ancestors. If **syncok()** is called with second argument **TRUE** then **wsyncup()** is called automatically whenever there is a change in the window.

The routine **wcursyncup()** updates the current cursor position of all the ancestors of the window to reflect the current cursor position of the window.

The routine **wsyncdown()** updates the status structure of the window to reflect the changes in the status structures of its ancestors. Applications seldom call this routine because it is called automatically by **wrefresh()**.

RETURN VALUES

Routines that return an integer return the integer **ERR** upon failure and an integer value other than **ERR** upon successful completion.

delwin() returns the integer **ERR** upon failure and **OK** upon successful completion.

Routines that return pointers return **NULL** on error.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

curs_refresh(3X), **curs_touch(3X)**, **curses(3X)**, **attributes(5)**

NOTES

The header `<curses.h>` automatically includes the headers `<stdio.h>` and `<unctrl.h>`.

If many small changes are made to the window, the **wsyncup()** option could degrade performance.

Note that **syncok()** may be a macro.

| NAME | cuserid – get character login name of the user | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <stdio.h> char *cuserid(char *s);</pre> | | | | |
| DESCRIPTION | <p>The cuserid() function generates a character-string representation of the login name under which the owner of the current process is logged in. If <i>s</i> is a null pointer, this representation is generated in an internal static area whose address is returned. Otherwise, <i>s</i> is assumed to point to an array of at least <code>L_cuserid</code> characters; the representation is left in this array. The constant <code>L_cuserid</code> is defined in the <code><stdio.h></code> header.</p> <p>In multithreaded applications, the caller must always supply an array <i>s</i> for the return value.</p> | | | | |
| RETURN VALUES | If the login name cannot be found, cuserid() returns a null pointer. If <i>s</i> is not a null pointer, the null character <code>'\0'</code> will be placed at <code>s[0]</code> . | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | getlogin(3C) , getpwnam(3C) , attributes(5) | | | | |

| | |
|-----------------|---|
| NAME | dbm, dbminit, dbmclose, fetch, store, delete, firstkey, nextkey – data base subroutines |
| SYNOPSIS | <pre>/usr/ucb/cc [<i>flag</i> ...] <i>file</i> ... -l dbm #include <dbm.h> typedef struct { char *dptr; int dsize; } datum; int dbminit(<i>file</i>); char * <i>file</i> ; int dbmclose(); datum fetch(<i>key</i>); datum <i>key</i> ; int store(<i>key</i>, <i>dat</i>); datum <i>key</i> , <i>dat</i> ; int delete(<i>key</i>); datum <i>key</i> ; datum firstkey() datum nextkey(<i>key</i>);</pre> |

| | |
|-----------------------------|---|
| <p>DESCRIPTION</p> | <pre>datum key ;</pre> <p>The dbm() library has been superseded by <code>ndbm</code> (see dbm_clearerr(3)).</p> <p>These functions maintain key/content pairs in a data base. The functions will handle very large (a billion blocks) databases and will access a keyed item in one or two file system accesses.</p> <p><i>key/dat</i> and their content are described by the <code>datum</code> typedef. A <code>datum</code> specifies a string of <i>dsize</i> bytes pointed to by <i>dptr</i>. Arbitrary binary data, as well as normal ASCII strings, are allowed. The data base is stored in two files. One file is a directory containing a bit map and has <code>.dir</code> as its suffix. The second file contains all data and has <code>.pag</code> as its suffix.</p> <p>Before a database can be accessed, it must be opened by dbmopen(). At the time of this call, the files <code>file.dir</code> and <code>file.pag</code> must exist. An empty database is created by creating zero-length <code>.dir</code> and <code>.pag</code> files.</p> <p>A database may be closed by calling dbmclose(). You must close a database before opening a new one.</p> <p>Once open, the data stored under a key is accessed by fetch() and data is placed under a key by store(). A key (and its associated contents) is deleted by delete(). A linear pass through all keys in a database may be made, in an (apparently) random order, by use of firstkey() and nextkey(). firstkey() will return the first key in the database. With any key nextkey() will return the next key in the database. This code will traverse the data base:</p> <pre>for (key = firstkey; key.dptr != NULL; key = nextkey(key))</pre> |
| <p>RETURN VALUES</p> | <p>All functions that return an <code>int</code> indicate errors with negative values. A zero return indicates no error. Routines that return a <code>datum</code> indicate errors with a <code>NULL (0) dptr</code>.</p> |
| <p>SEE ALSO</p> | <p><code>ar(1)</code>, <code>cat(1)</code>, <code>cp(1)</code>, <code>tar(1)</code>, dbm_clearerr(3)</p> |
| <p>NOTES</p> | <p>Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.</p> <p>The <code>.pag</code> file will contain holes so that its apparent size may be larger than its actual content. Older versions of the UNIX operating system may create real</p> |

file blocks for these holes when touched. These files cannot be copied by normal means (`cp(1)` , `cat(1)` , `tar(1)` , `ar(1)`)without filling in the holes.

dptr pointers returned by these subroutines point into static storage that is changed by subsequent calls.

The sum of the sizes of a key/content pair must not exceed the internal block size (currently 1024 bytes). Moreover all key/content pairs that hash together must fit on a single block. `store` will return an error in the event that a disk block fills with inseparable data.

`delete()` does not physically reclaim file space, although it does make it available for reuse.

The order of keys presented by `firstkey()` and `nextkey()` depends on a hashing function, not on anything interesting.

There are no interlocks and no reliable cache flushing; thus concurrent updating and reading is risky.

The database files (`file.dir` and `file.pag`)are binary and are architecture-specific (for example, they depend on the architecture's byte order.) These files are not guaranteed to be portable across architectures.

| | |
|--------------------|--|
| NAME | dbm_clearerr, dbm_close, dbm_delete, dbm_error, dbm_fetch, dbm_firstkey, dbm_nextkey, dbm_open, dbm_store – database functions |
| SYNOPSIS | <pre>#include <ndbm.h> int dbm_clearerr(DBM * db); void dbm_close(DBM * db); int dbm_delete(DBM * db, datum key); int dbm_error(DBM * db); datum dbm_fetch(DBM * db, datum key); datum dbm_firstkey(DBM * db); datum dbm_nextkey(DBM * db); DBM * dbm_open(const char * file, int open_flags, mode_t file_mode); int dbm_store(DBM * db, datum key, datum content, int store_mode);</pre> |
| DESCRIPTION | <p>These functions create, access and modify a database. They maintain <i>key</i> / <i>content</i> pairs in a database. The functions will handle large databases (up to a billion blocks) and will access a keyed item in one or two file system accesses. This package replaces the earlier dbm(3B) library, which managed only a single database.</p> <p><i>key</i> s and <i>content</i> s are described by the <code>datum</code> typedef. A <code>datum</code> consists of at least two members, <code>dptr</code> and <code>dsize</code> . The <code>dptr</code> member points to an object that is <code>dsize</code> bytes in length. Arbitrary binary data, as well as ASCII character strings, may be stored in the object pointed to by <code>dptr</code> .</p> <p>The database is stored in two files. One file is a directory containing a bit map of keys and has <code>.dir</code> as its suffix. The second file contains all data and has <code>.pag</code> as its suffix.</p> <p>The dbm_open() function opens a database. The <code>file</code> argument to the function is the pathname of the database. The function opens two files named <code>file.dir</code> and <code>file.pag</code> . The <code>open_flags</code> argument has the same meaning as the <code>flags</code> argument of <code>open(2)</code> except that a database opened for write-only access opens the files for read and write access. The <code>file_mode</code> argument has the same meaning as the third argument of <code>open(2)</code> .</p> <p>The dbm_close() function closes a database. The argument <code>db</code> must be a pointer to a <code>dbm</code> structure that has been returned from a call to dbm_open() .</p> <p>The dbm_fetch() function reads a record from a database. The argument <code>db</code> is a pointer to a database structure that has been returned from a call to</p> |

dbm_open() . The argument *key* is a datum that has been initialized by the application program to the value of the key that matches the key of the record the program is fetching.

The **dbm_store()** function writes a record to a database. The argument *db* is a pointer to a database structure that has been returned from a call to **dbm_open()** . The argument *key* is a datum that has been initialized by the application program to the value of the key that identifies (for subsequent reading, writing or deleting) the record the program is writing. The argument *content* is a datum that has been initialized by the application program to the value of the record the program is writing. The argument *store_mode* controls whether **dbm_store()** replaces any pre-existing record that has the same key that is specified by the *key* argument. The application program must set *store_mode* to either `DBM_INSERT` or `DBM_REPLACE` . If the database contains a record that matches the *key* argument and *store_mode* is `DBM_REPLACE` , the existing record is replaced with the new record. If the database contains a record that matches the *key* argument and *store_mode* is `DBM_INSERT` , the existing record is not replaced with the new record. If the database does not contain a record that matches the *key* argument and *store_mode* is either `DBM_INSERT` or `DBM_REPLACE` , the new record is inserted in the database.

The **dbm_delete()** function deletes a record and its key from the database. The argument *db* is a pointer to a database structure that has been returned from a call to **dbm_open()** . The argument *key* is a datum that has been initialized by the application program to the value of the key that identifies the record the program is deleting.

The **dbm_firstkey()** function returns the first key in the database. The argument *db* is a pointer to a database structure that has been returned from a call to **dbm_open()** .

The **dbm_nextkey()** function returns the next key in the database. The argument *db* is a pointer to a database structure that has been returned from a call to **dbm_open()** . The **dbm_firstkey()** function must be called before calling **dbm_nextkey()** . Subsequent calls to **dbm_nextkey()** return the next key until all of the keys in the database have been returned.

The **dbm_error()** function returns the error condition of the database. The argument *db* is a pointer to a database structure that has been returned from a call to **dbm_open()** .

The **dbm_clearerr()** function clears the error condition of the database. The argument *db* is a pointer to a database structure that has been returned from a call to **dbm_open()** .

These database functions support key/content pairs of at least 1024 bytes.

RETURN VALUES

The **dbm_store()** and **dbm_delete()** functions return 0 when they succeed and a negative value when they fail.

The **dbm_store()** function returns 1 if it is called with a *flags* value of `DBM_INSERT` and the function finds an existing record with the same key.

The **dbm_error()** function returns 0 if the error condition is not set and returns a non-zero value if the error condition is set.

The return value of **dbm_clearerr()** is unspecified .

The **dbm_firstkey()** and **dbm_nextkey()** functions return a `datum` . When the end of the database is reached, the `dptr` member of the key is a null pointer. If an error is detected, the `dptr` member of the key is a null pointer and the error condition of the database is set.

The **dbm_fetch()** function returns a content `datum` . If no record in the database matches the key or if an error condition has been detected in the database, the `dptr` member of the content is a null pointer.

The **dbm_open()** function returns a pointer to a database structure. If an error is detected during the operation, **dbm_open()** returns a `(DBM *)0`.

ERRORS

No errors are defined.

USAGE

The following code can be used to traverse the database:

```
for(key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))
```

The `dbm_` functions provided in this library should not be confused in any way with those of a general-purpose database management system. These functions do not provide for multiple search keys per entry, they do not protect against multi-user access (in other words they do not lock records or files), and they do not provide the many other useful database functions that are found in more robust database management systems. Creating and updating databases by use of these functions is relatively slow because of data copies that occur upon hash collisions. These functions are useful for applications requiring fast lookup of relatively static information that is to be indexed by a single key.

The `dptr` pointers returned by these functions may point into static storage that may be changed by subsequent calls.

The **dbm_delete()** function does not physically reclaim file space, although it does make it available for reuse.

After calling **dbm_store()** or **dbm_delete()** during a pass through the keys by **dbm_firstkey()** and **dbm_nextkey()** , the application should reset the database by calling **dbm_firstkey()** before again calling **dbm_nextkey()** .

EXAMPLES**EXAMPLE 1** Using the Database Functions

The following example stores and retrieves a phone number, using the name as the key. Note that this example does not include error checking.

```
#include <ndbm.h>
#include <stdio.h>
#include <fcntl.h>
#define NAME      "Bill"
#define PHONE_NO  "123-4567"
#define DB_NAME   "phones"
main()
{
    DBM *db;
    datum name = {NAME, sizeof (NAME)};
    datum put_phone_no = {PHONE_NO, sizeof (PHONE_NO)};
    datum get_phone_no;
    /* Open the database and store the record */
    db = dbm_open(DB_NAME, O_RDWR | O_CREAT, 0660);
    (void) dbm_store(db, name, put_phone_no, DBM_INSERT);
    /* Retrieve the record */
    get_phone_no = dbm_fetch(db, name);
    (void) printf("Name: %s, Phone Number: %s\n",
        name.dptr,
        get_phone_no.dptr);
    /* Close the database */
    dbm_close(db);
    return (0);
}
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

ar(1), **cat(1)**, **cp(1)**, **tar(1)**, **open(2)**, **dbm(3B)**, **netconfig(4)**, **attributes(5)**

NOTES

The `.pag` file will contain holes so that its apparent size may be larger than its actual content. Older versions of the UNIX operating system may create real file blocks for these holes when touched. These files cannot be copied by normal means (**cp(1)**, **cat(1)**, **tar(1)**, **ar(1)**) without filling in the holes.

The sum of the sizes of a *key / content* pair must not exceed the internal block size (currently 1024 bytes). Moreover all *key / content* pairs that hash together must fit on a single block. **dbm_store()** will return an error in the event that a disk block fills with inseparable data.

The order of keys presented by **dbm_firstkey()** and **dbm_nextkey()** depends on a hashing function.

There are no interlocks and no reliable cache flushing; thus concurrent updating and reading is risky.

The database files (`file.dir` and `file.pag`) are binary and are architecture-specific (for example, they depend on the architecture's byte order.) These files are not guaranteed to be portable across architectures.

| | |
|--------------------|---|
| NAME | decimal_to_floating, decimal_to_single, decimal_to_double, decimal_to_extended, decimal_to_quadruple – convert decimal record to floating-point value |
| SYNOPSIS | <pre>#include <floatingpoint.h> void decimal_to_single(single * px, decimal_mode * pm, decimal_record * pd, fp_exception_field_type * ps); void decimal_to_double(double * px, decimal_mode * pm, decimal_record * pd, fp_exception_field_type * ps); void decimal_to_extended(extended * px, decimal_mode * pm, decimal_record * pd, fp_exception_field_type * ps); void decimal_to_quadruple(quadruple * px, decimal_mode * pm, decimal_record * pd, fp_exception_field_type * ps);</pre> |
| DESCRIPTION | <p>The decimal_to_floating() functions convert the decimal record at <i>*pd</i> into a floating-point value at <i>*px</i>, observing the modes specified in <i>*pm</i> and setting exceptions in <i>*ps</i>. If there are no IEEE exceptions, <i>*ps</i> will be zero.</p> <p><i>pd->sign</i> and <i>pd->fpclass</i> are always taken into account. <i>pd->exponent</i>, <i>pd->ds</i> and <i>pd->ndigits</i> are used when <i>pd->fpclass</i> is <i>fp_normal</i> or <i>fp_subnormal</i>. In these cases <i>pd->ds</i> must contain one or more ascii digits followed by a NULL and <i>pd->ndigits</i> is assumed to be the length of the string <i>pd->ds</i>. Notice that for efficiency reasons, the assumption that <i>pd->ndigits</i> == strlen(<i>pd->ds</i>) is NEVER verified.</p> <p>On output, <i>*px</i> is set to a correctly rounded approximation to</p> $(pd->sign)*(pd->ds)*10^{(pd->exponent)}$ <p>Thus if <i>pd->exponent</i> == -2 and <i>pd->ds</i> == "1234", <i>*px</i> will get 12.34 rounded to storage precision. <i>pd->ds</i> cannot have more than DECIMAL_STRING_LENGTH-1 significant digits because one character is used to terminate the string with a NULL. If <i>pd->more</i> != 0 on input then additional nonzero digits follow those in <i>pd->ds</i>; <i>fp_inexact</i> is set accordingly on output in <i>*ps</i>.</p> <p>* <i>px</i> is correctly rounded according to the IEEE rounding modes in <i>pm->rd</i>. <i>*ps</i> is set to contain <i>fp_inexact</i>, <i>fp_underflow</i>, or <i>fp_overflow</i> if any of these arise.</p> <p><i>pm->df</i> and <i>pm->ndigits</i> are not used.</p> |

`strtod(3C)` , `scanf(3S)` , `fscanf(3S)` , and `sscanf(3S)` all use `decimal_to_double()` .

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`fscanf(3S)` , `scanf(3S)` , `sscanf(3S)` , `strtod(3C)` , `attributes(5)`

| | |
|----------------------|--|
| NAME | def_prog_mode, def_shell_mode, reset_prog_mode, reset_shell_mode – save/restore terminal modes |
| SYNOPSIS | <pre>#include <curses.h> int def_prog_mode(void); int def_shell_mode(void); int reset_prog_mode(void); int reset_shell_mode(void);</pre> |
| DESCRIPTION | <p>The def_prog_mode() and def_shell_mode() functions save the current terminal modes as "program" (within X/Open Curses) or "shell" (outside X/Open Curses). The modes are saved automatically by initscr(3XC), newterm(3XC), and setupterm(3XC).</p> <p>The reset_prog_mode() and reset_shell_mode() functions reset the current terminal modes to "program" (within X/Open Curses) or "shell" (outside X/Open Curses). The endwin(3XC) function automatically calls the reset_shell_mode() function and the doupdate(3XC) function calls the reset_prog_mode() function after calling endwin().</p> |
| RETURN VALUES | On success, these functions return OK . Otherwise, they return ERR . |
| ERRORS | None. |
| SEE ALSO | endwin(3XC) , initscr(3XC) , newterm(3XC) , setupterm(3XC) |

| | |
|----------------------|---|
| NAME | delay_output – delays output |
| SYNOPSIS | <pre>#include <curses.h> int delay_output(int ms);</pre> |
| PARAMETERS | ms Is the number of milliseconds to delay the output. |
| DESCRIPTION | The delay_output() function delays output for <i>ms</i> milliseconds by inserting pad characters in the output stream. |
| RETURN VALUES | On success, the delay_output() function returns OK. Otherwise, it returns ERR. |
| ERRORS | None. |
| SEE ALSO | napms(3XC) |

| | |
|----------------------|---|
| NAME | delch, mvdelch, mvwdelch, wdelch – remove a character |
| SYNOPSIS | <pre>#include <curses.h> int delch(void); int mvdelch(int y, int x); int mvwdelch(WINDOW * win, int y, int x); int wdelch(WINDOW * win);</pre> |
| PARAMETERS | <p>y Is the y (row) coordinate of the position of the character to be removed.</p> <p>x Is the x (column) coordinate of the position of the character to be removed.</p> <p>win Is a pointer to the window containing the character to be removed.</p> |
| DESCRIPTION | <p>The delch() and wdelch() functions delete the character at the current cursor position from <code>stdscr</code> and <code>win</code>, respectively. All remaining characters after cursor through to the end of the line are shifted one character towards the start of the line. The last character on the line becomes a space; characters on other lines are not affected.</p> <p>The mvdelch() and mvwdelch() functions delete the character at the position specified by the <code>x</code> and <code>y</code> parameters; the former deletes the character from <code>stdscr</code>; the latter from <code>win</code>.</p> |
| RETURN VALUES | On success, these functions return <code>OK</code> . Otherwise, they return <code>ERR</code> . |
| ERRORS | None. |
| SEE ALSO | <code>bkgdset(3XC)</code> , <code>insch(3XC)</code> |

| | |
|--------------------|---|
| NAME | del_curterm, restartterm, set_curterm, setterm, setupterm – free space pointed to by terminal |
| SYNOPSIS | <pre>#include <term.h> int del_curterm(TERMINAL * oterm); int restartterm(char * term, int fildes, int * errret); TERMINAL *set_curterm (TERMINAL * nterm); int setterm(char * term); int setupterm(char * term, int fildes, int * errret);</pre> |
| PARAMETERS | <p>oterm Is the terminal type for which to free space.</p> <p>term Is the terminal type for which variables are set.</p> <p>fildes Is a file descriptor initialized for output.</p> <p>errret Is a pointer to an integer in which the status value is stored.</p> <p>nterm Is the new terminal to become the current terminal.</p> |
| DESCRIPTION | <p>Within X/Open Curses, the setupterm() function is automatically called by the <code>initscr (3XC)</code> and <code>newterm (3XC)</code> functions. This function can be also be used outside of X/Open Curses when a program has to deal directly with the <code>terminfo</code> database to handle certain terminal capabilities. The use of appropriate X/Open Curses functions is recommended in all other situations.</p> <p>The setupterm() function loads terminal-dependent variables for the <code>terminfo</code> layer of X/Open Curses. The setupterm() function initializes the <code>terminfo</code> variables <code>lines</code> and <code>columns</code> such that if <code>use_env(FALSE)</code> has been called, the <code>terminfo</code> values assigned in the database are used regardless of the environmental variables <code>LINES</code> and <code>COLUMNS</code> or the program's window dimensions; when <code>use_env(TRUE)</code> has been called, which is the default, the environment variables <code>LINES</code> and <code>COLUMNS</code> are used, if they exist. If the environment variables do not exist and the program is running in a window, the current window size is used.</p> <p>The <i>term</i> parameter of setupterm() specifies the terminal; if null, terminal type is taken from the <code>TERM</code> environment variable. All output is sent to <i>fildes</i> which is initialized for output. If <i>errret</i> is not null, <code>OK</code> or <code>ERR</code> is returned and a status value is stored in the integer pointed to by <i>errret</i> . The following status values may be returned:</p> |

| Value | Description |
|-------|--------------------------------------|
| 1 | Normal |
| 0 | Terminal could not be found |
| -1 | terminfo database could not be found |

If *errret* is null, an error message is printed, and the **setupterm()** function calls the **exit()** function with a non-zero parameter.

The **setterm()** macro is an older version of **setupterm()**. It is included for compatibility with previous versions of Curses. New programs should use **setupterm()**.

The **set_curterm()** function sets the *cur_term* variable to *nterm*. The values from *nterm* as well as other state information for the terminal are used by X/Open Curses functions such as **beep(3XC)**, **flash(3XC)**, **mvcur(3XC)**, **tigetflag(3XC)**, **tigetstr(3XC)**, and **tigetnum(3XC)**.

The **del_curterm()** function frees the space pointed to by *oterm*. If *oterm* and the *cur_term* variable are the same, all Boolean, numeric, or string *terminfo* variables will refer to invalid memory locations until you call **setupterm()** and specify a new terminal type.

The **restartterm()** function assumes that a call to **setupterm()** has already been made (probably from **initscr()** or **newterm()**). It allows you to specify a new terminal type in *term* and updates the data returned by **baudrate(3XC)** based on *fildev*. Other information created by the **initscr()**, **newterm()**, and **setupterm()** functions is preserved.

RETURN VALUES

On success, the **set_curterm()** function returns the previous value of *cur_term*. Otherwise, it returns a null pointer.

On success, the other functions return **OK**. Otherwise, they return **ERR**.

ERRORS

None.

SEE ALSO

baudrate(3XC), **beep(3XC)**, **initscr(3XC)**, **mvcur(3XC)**, **tigetflag(3XC)**, **use_env(3XC)**

| | |
|----------------------|---|
| NAME | deleteln, wdeleteln – remove a line |
| SYNOPSIS | <pre>#include <curses.h> int deleteln(void); int wdeleteln(WINDOW * win);</pre> |
| PARAMETERS | <p>win Is a pointer to the window from which the line is removed.</p> |
| DESCRIPTION | The deleteln() and wdeleteln() functions delete the line containing the cursor from stdscr and win , respectively. All lines below the one deleted are moved up one line. The last line of the window becomes blank. The position of the cursor is unchanged. |
| RETURN VALUES | On success, these functions return OK . Otherwise, they return ERR . |
| ERRORS | None. |
| SEE ALSO | bkgdset(3XC) , insdelln(3XC) , insertln(3XC) |

| | |
|----------------------|--|
| NAME | delscreen – free space associated with the SCREEN data structure |
| SYNOPSIS | <pre>#include <curses.h> void delscreen(SCREEN *sp);</pre> |
| PARAMETERS | <p>sp Is a pointer to the screen structure for which to free space.</p> |
| DESCRIPTION | The delscreen() function frees space associated with the SCREEN data structure. This function should be called after endwin(3XC) if a SCREEN data structure is no longer needed. |
| RETURN VALUES | The delscreen() function does not return a value. |
| ERRORS | None. |
| SEE ALSO | endwin(3XC) , initscr(3XC) , newterm(3XC) |

| | |
|----------------------|---|
| NAME | delwin – delete a window |
| SYNOPSIS | <pre>#include <curses.h> int delwin(WINDOW *win);</pre> |
| PARAMETERS | win Is a pointer to the window that is to be deleted. |
| DESCRIPTION | <p>The delwin() function deletes the specified window, freeing up the memory associated with it.</p> <p>Deleting a parent window without deleting its subwindows and then trying to manipulate the subwindows will have undefined results.</p> |
| RETURN VALUES | On success, this functions returns OK. Otherwise, it returns ERR. |
| ERRORS | None. |
| SEE ALSO | derwin(3XC) , dupwin(3XC) |

| | |
|----------------------|---|
| NAME | demangle, cplus_demangle – decode a C++ encoded symbol name |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> [<i>library</i> ...] -l demangle</pre> <pre>#include <demangle.h></pre> <pre>int cplus_demangle(const char * <i>symbol</i>, char * <i>prototype</i>, size_t <i>size</i>);</pre> |
| DESCRIPTION | <p>The cplus_demangle() function decodes (demangles) a C++ linker symbol name (mangled name) into a (partial) C++ prototype, if possible. C++ mangled names may not have enough information to form a complete prototype.</p> <p>The <i>symbol</i> string argument points to the input mangled name.</p> <p>The <i>prototype</i> argument points to a user-specified output string buffer, of <i>size</i> bytes.</p> <p>The cplus_demangle() function operates on mangled names generated by SPARCCompilers C++ 3.0.1, 4.0.1, 4.1 and 4.2.</p> <p>The cplus_demangle() function improves and replaces the demangle() function.</p> <p>Refer to the <code>CC.1</code>, <code>dem.1</code>, and <code>c++filt.1</code> manual pages in the <code>/opt/SUNWspro/man/man1</code> directory. These pages are only available with the SPROcc package.</p> |
| RETURN VALUES | <p>The cplus_demangle() function returns the following values:</p> <p>0 The <i>symbol</i> argument is a valid mangled name and <i>prototype</i> contains a (partial) prototype for the symbol.</p> <p>DEMANGLE_ENAME The <i>symbol</i> argument is not a valid mangled name and the content of <i>prototype</i> is a copy of the symbol.</p> |

DEMANGLE_ESPACE

The *prototype* output buffer is too small to contain the prototype (or the symbol), and the content of *prototype* is undefined.

| | |
|--------------------|---|
| NAME | derwin, newwin, subwin – create a new window or subwindow |
| SYNOPSIS | <pre>#include <curses.h> WINDOW * derwin(WINDOW * orig, int nlines, int ncols, int begin_y, int begin_x); WINDOW * newwin(int nlines, int ncols, int begin_y, int begin_x); WINDOW * subwin(WINDOW * orig, int nlines, int ncols, int begin_y, int begin_x);</pre> |
| PARAMETERS | <p>orig Is a pointer to the parent window for the newly created subwindow.</p> <p>nlines Is the number of lines in the subwindow.</p> <p>ncols Is the number of columns in the subwindow.</p> <p>begin_y Is the y (row) coordinate of the upper left corner of the subwindow, relative to the parent window.</p> <p>begin_x Is the x (column) coordinate of the upper left corner of the subwindow, relative to the parent window.</p> |
| DESCRIPTION | <p>The derwin() function creates a subwindow within window <i>orig</i> , with the specified number of lines and columns, and upper left corner positioned at <i>begin_x</i> , <i>begin_y</i> relative to window <i>orig</i> . A pointer to the new window structure is returned.</p> <p>The newwin() function creates a new window with the specified number of lines and columns and upper left corner positioned at <i>begin_x</i> , <i>begin_y</i> . A pointer to the new window structure is returned. A full-screen window can be created by calling <code>newwin(0,0,0,0)</code> .</p> <p>If the number of lines specified is zero, newwin() uses a default value of <code>LINES</code> minus <i>begin_y</i> ; if the number of columns specified is zero, newwin() uses the default value of <code>COLS</code> minus <i>begin_x</i> .</p> <p>The subwin() function creates a subwindow within window <i>orig</i> , with the specified number of lines and columns, and upper left corner positioned at <i>begin_x</i> , <i>begin_y</i> (relative to the physical screen, <i>not</i> to window <i>orig</i>). A pointer to the new window structure is returned.</p> <p>The original window and subwindow share character storage of the overlapping area (each window maintains its own pointers, cursor location, and other items). This means that characters and attributes are identical in overlapping areas regardless of which window characters are written to.</p> |

When using subwindows, it is often necessary to call `touchwin(3XC)` before `wrefresh(3XC)` to maintain proper screen contents.

RETURN VALUES

On success, these functions return a pointer to the newly-created window. Otherwise, they return `ERR`.

ERRORS

None.

SEE ALSO

`doudate(3XC)`, `is_linetouched(3XC)`

| | | | | | | | | | |
|----------------------|---|-------------|-----------|-------------------|---|----------------|--|-----------------|---------------------------|
| NAME | des_crypt, ecb_crypt, cbc_crypt, des_setparity, DES_FAILED – fast DES encryption | | | | | | | | |
| SYNOPSIS | <pre>#include <rpc/des_crypt.h> int ecb_crypt(char * key, char * data, unsigned datalen, unsigned mode); int cbc_crypt(char * key, char * data, unsigned datalen, unsigned mode, char * ivec); void des_setparity(char * key); int DES_FAILED(int stat);</pre> | | | | | | | | |
| DESCRIPTION | <p>ecb_crypt() and cbc_crypt() implement the NBS DES (Data Encryption Standard). These routines are faster and more general purpose than crypt(3C). They also are able to utilize DES hardware if it is available. ecb_crypt() encrypts in ECB (Electronic Code Book) mode, which encrypts blocks of data independently. cbc_crypt() encrypts in CBC (Cipher Block Chaining) mode, which chains together successive blocks. CBC mode protects against insertions, deletions, and substitutions of blocks. Also, regularities in the clear text will not appear in the cipher text.</p> <p>The first parameter, <i>key</i>, is the 8-byte encryption key with parity. To set the key's parity, which for DES is in the low bit of each byte, use des_setparity(). The second parameter, <i>data</i>, contains the data to be encrypted or decrypted. The third parameter, <i>datalen</i>, is the length in bytes of <i>data</i>, which must be a multiple of 8. The fourth parameter, <i>mode</i>, is formed by OR'ing together the DES_ENCRYPT or DES_DECRYPT to specify the encryption direction and DES_HW or DES_SW to specify software or hardware encryption. If DES_HW is specified, and there is no hardware, then the encryption is performed in software and the routine returns DESERR_NOHWDEVICE.</p> <p>For cbc_crypt(), the parameter <i>ivec</i> is the 8-byte initialization vector for the chaining. It is updated to the next initialization vector upon successful return.</p> | | | | | | | | |
| RETURN VALUES | <p>Given a result status <i>stat</i>, the macro DES_FAILED is false only for the first two statuses.</p> <table border="0"> <tr> <td style="padding-right: 20px;">DESERR_NONE</td> <td>No error.</td> </tr> <tr> <td>DESERR_NOHWDEVICE</td> <td>Encryption succeeded, but done in software instead of the requested hardware.</td> </tr> <tr> <td>DESERR_HWERROR</td> <td>An error occurred in the hardware or driver.</td> </tr> <tr> <td>DESERR_BADPARAM</td> <td>Bad parameter to routine.</td> </tr> </table> | DESERR_NONE | No error. | DESERR_NOHWDEVICE | Encryption succeeded, but done in software instead of the requested hardware. | DESERR_HWERROR | An error occurred in the hardware or driver. | DESERR_BADPARAM | Bad parameter to routine. |
| DESERR_NONE | No error. | | | | | | | | |
| DESERR_NOHWDEVICE | Encryption succeeded, but done in software instead of the requested hardware. | | | | | | | | |
| DESERR_HWERROR | An error occurred in the hardware or driver. | | | | | | | | |
| DESERR_BADPARAM | Bad parameter to routine. | | | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | | | | | |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| Availability | SUNWcry |
| MT-Level | MT-Safe |

SEE ALSO `crypt(3C)` , `attributes(5)`

RESTRICTIONS This program is not available on software shipped outside the U.S. These routines are available only with the U.S. Encryption kit.

NOTES When compiling multi-thread applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multi-thread applications.

| | |
|--------------------|--|
| NAME | devid_get, devid_free, devid_get_minor_name, devid_deviceid_to_nmlist, devid_free_nmlist, devid_compare, devid_sizeof – device id interfaces for user applications |
| SYNOPSIS | <pre>#include <devid.h> int devid_get(int fd, ddi_devid_t * retdevid); void devid_free(ddi_devid_t devid); int devid_get_minor_name(int fd, char ** retminor_name); int devid_deviceid_to_nmlist(char * search_path, ddi_devid_t devid, char * minor_name, devid_nmlist_t ** retlist); void devid_free_nmlist(devid_nmlist_t * list); int devid_compare(ddi_devid_t devid1, ddi_devid_t devid2); size_t devid_sizeof(ddi_devid_t devid);</pre> |
| DESCRIPTION | <p>The following routines are used to provide unique identifiers, device ids, for devices. Specifically, applications and device drivers use these interfaces to identify and locate devices, independent of the device's physical connection or its logical device name or number.</p> <p>devid_get() returns the device id, in <i>retdevid</i> , for the device associated with the open file descriptor <i>fd</i> , which refers to any device. If the device does not have a device id associated with it then an error is returned. The caller of this function must free the memory allocated for the <i>retdevid</i> returned, using the devid_free() function.</p> <p>devid_free() frees the allocated space for the passed-in <i>devid</i> , allocated by devid_get() .</p> <p>devid_get_minor_name() returns the minor name, in <i>retminor_name</i> , for the device associated with the open file descriptor <i>fd</i> . This name is specific to the particular minor number, but is "instance number" specific. The caller of this function must free the memory allocated for the returned string in <i>retminor_name</i> , using the devid_free() function.</p> <p>devid_deviceid_to_nmlist() returns an array of <i>devid_nmlist</i> structures, where each entry matches the devid id and minor name passed in. The <i>devid_nmlist</i> structure contains the device name and device number. The last entry of the array has a null pointer for the <i>devname</i> and <code>NODEV</code> for the device number.</p> <p>This function walks through the file tree, starting at <i>search_path</i> . For each device with a matching device id and minor name tuple, a device name and device number are added to the <i>retlist</i> . If no matches are found, an error is</p> |

returned. The caller of this function must free the memory allocated for the returned array with the **devid_free_nmlist()** function.

devid_free_nmlist() frees the memory allocated by the **devid_deviceid_to_nmlist()** function.

devid_compare() compares two device ids byte-by-byte and determines both equality and sort order. The function returns an integer greater than zero if the device id pointed to by *devid1* is greater than the device id pointed to by *devid2*. It returns zero if the device id pointed to by *devid1* is equal to the device id pointed to by *devid2*. It returns an integer less than zero if the device id pointed to by *devid1* is less than the device id pointed to by *devid2*.

devid_sizeof() returns the size in number of bytes allocated for the *devid*.

RETURN VALUES

The following functions return 0 upon successful completion: **devid_get()**, **devid_get_minor_name()**, and **devid_deviceid_to_nmlist()**.

Otherwise, -1 is returned and `errno` is set to indicate the error.

The function **devid_compare()** returns the following values:

≤-1 The device id pointed to by *devid1* is less than the device id pointed to by *devid2*.

0 The device id pointed to by *devid1* is equal to the device id pointed to by *devid2*.

≥1 The device id pointed to by *devid1* is greater than the device id pointed to by *devid2*.

The return value from **devid_sizeof()** is the size in number of bytes allocated for the *devid*.

EXAMPLES

EXAMPLE 1 Using the **devid_get()** and **devid_get_minor_name()** Functions

The following example shows the proper use of **devid_get()** and **devid_get_minor_name()** to free the space allocated for the *device id* and *minor name*.

```
int fd;
\011ddi_devid_t\011devid;
\011char\011\011*minor_name;
\011if ((fd = open("/dev/dsk/c0t3d0s0", O_RDONLY|O_NDELAY)) < 0) {
\011\011...
\011}
\011if (devid_get(fd, &devid) != 0) {
\011\011...
\011}
\011if (devid_get_minor_name(fd, &minor_name) != 0) {
\011\011...
\011}
\011< process devid and minor_name >
```

```
\011devid_free(devid);
\011free(minor_name);
```

EXAMPLE 2 Using the **devid_deviceid_to_nmlist()** and **devid_free_nmlist()** Functions

The following example shows the proper use of **devid_deviceid_to_nmlist()** and **devid_free_nmlist()** :

```
devid_nmlist_t *list = NULL;
\011int\011err;
    err = devid_deviceid_to_nmlist("/dev/rdisk", devid, minor_name, &list);
\011if (err)
\011\011return (err);
\011< loop through list and process device names and device numbers >
    devid_free_nmlist(list);
```

FILES

| | |
|------------------------|---|
| /usr/lib/libdevid.so.1 | The location of the device id library interfaces. |
| /usr/lib/libdevid.so | A symlink to /usr/lib/libdevid.so.1. |

ATTRIBUTES

See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | MT-Safe |

SEE ALSO

libdevid(4), **attributes(5)**, **ddi_devid_devlist(9F)**, **ddi_devid_free(9F)**, **ddi_devid_init(9F)**, **ddi_devid_register(9F)**, **ddi_devid_sizeof(9F)**, **ddi_devid_unregister(9F)**, **ddi_devid_valid(9F)**

| | | | | | | | |
|--------------------|--|-----|----------|---|--------|---|--------|
| NAME | dial – establish an outgoing terminal line connection | | | | | | |
| SYNOPSIS | <pre>cc [flag ...] file ... -lnsl [library ...] #include <dial.h></pre> <pre>int dial(CALL call);</pre> <pre>void undial(int fd);</pre> | | | | | | |
| DESCRIPTION | <p>dial() returns a file-descriptor for a terminal line open for read/write. The argument to dial() is a CALL structure (defined in the header <code><dial.h></code>).</p> <p>When finished with the terminal line, the calling program must invoke undial() to release the semaphore that has been set during the allocation of the terminal device.</p> <p>CALL is defined in the header <code><dial.h></code> and has the following members:</p> <pre>struct termio *attr; /* pointer to termio attribute struct */ int baud; /* transmission data rate */ int speed; /* 212A modem: low=300, high=1200 */ char *line; /* device name for out-going line */ char *telno; /* pointer to tel-no digits string */ int modem; /* specify modem control for direct lines */ char *device; /* unused */ int dev_len; /* unused */</pre> <p>The CALL element <code>speed</code> is intended only for use with an outgoing dialed call, in which case its value should be the desired transmission baud rate. The CALL element <code>baud</code> is no longer used.</p> <p>If the desired terminal line is a direct line, a string pointer to its device-name should be placed in the <code>line</code> element in the CALL structure. Legal values for such terminal device names are kept in the <code>Devices</code> file. In this case, the value of the <code>baud</code> element should be set to -1. This value will cause <code>dial</code> to determine the correct value from the <code><Devices></code> file.</p> <p>The <code>telno</code> element is for a pointer to a character string representing the telephone number to be dialed. Such numbers may consist only of these characters:</p> <table border="0" style="margin-left: 20px;"> <tr> <td style="padding-right: 40px;">0-9</td> <td>dial 0-9</td> </tr> <tr> <td>*</td> <td>dail *</td> </tr> <tr> <td>#</td> <td>dail #</td> </tr> </table> | 0-9 | dial 0-9 | * | dail * | # | dail # |
| 0-9 | dial 0-9 | | | | | | |
| * | dail * | | | | | | |
| # | dail # | | | | | | |

```

=          wait for secondary dial tone
-          delay for approximately 4 seconds

```

The CALL element `modem` is used to specify modem control for direct lines. This element should be non-zero if modem control is required. The CALL element `attr` is a pointer to a `termio` structure, as defined in the header `<termio.h>`. A NULL value for this pointer element may be passed to the `dial` function, but if such a structure is included, the elements specified in it will be set for the outgoing terminal line before the connection is established. This setting is often important for certain attributes such as parity and baud-rate.

The CALL elements `device` and `dev_len` are no longer used. They are retained in the CALL structure for compatibility reasons.

RETURN VALUES

On failure, a negative value indicating the reason for the failure will be returned. Mnemonics for these negative indices as listed here are defined in the header `<dial.h>`.

```

INTRPT  -1      /* interrupt occurred */
D_HUNG  -2      /* dialer hung (no return from write) */
NO_ANS  -3      /* no answer within 10 seconds */
ILL_BD  -4      /* illegal baud-rate */
A_PROB  -5      /* acu problem (open() failure) */
L_PROB  -6      /* line problem (open() failure) */
NO_LdV  -7      /* can't open Devices file */
DV_NT_A -8      /* requested device not available */
DV_NT_K -9      /* requested device not known */
NO_BD_A -10     /* no device available at requested baud */
NO_BD_K -11     /* no device known at requested baud */
DV_NT_E -12     /* requested speed does not match */
BAD_SYS -13     /* system not in Systems file*/

```

FILES

```

/etc/uucp/Devices
/etc/uucp/Systems
/var/spool/uucp/LCK..tty-device

```

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `uucp(1C)`, `alarm(2)`, `read(2)`, `write(2)`, `attributes(5)`, `termio(7I)`

NOTES

Including the header `<dial.h>` automatically includes the header `<termio.h>`. An `alarm(2)` system call for 3600 seconds is made (and caught) within the `dial` module for the purpose of “touching” the `LCK..` file and constitutes the device allocation semaphore for the terminal device. Otherwise, `uucp(1C)` may simply delete the `LCK..` entry on its 90-minute clean-up rounds. The alarm may go off while the user program is in a `read(2)` or `write(2)` function, causing an apparent error return. If the user program expects to be around for an hour or more, error returns from `read(s)` should be checked for (`errno==EINTR`), and the `read()` possibly reissued.

This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

| | |
|-----------------------|--|
| NAME | di_binding_name, di_bus_addr, di_compatible_names, di_devid, di_driver_name, di_driver_ops, di_instance, di_nodeid, di_node_name – return libdevinfo node information |
| SYNOPSIS | <pre>char * di_binding_name(di_node_t node); char * di_bus_addr(di_node_t node); int di_compatible_names(di_node_t node, char ** names); ddi_devid_t di_devid(di_node_t node); char * di_driver_name(di_node_t node); uint_t di_driver_ops(di_node_t node); int di_instance(di_node_t node); int di_nodeid(di_node_t node); char * di_node_name(di_node_t node);</pre> |
| DESCRIPTION | These interfaces are used to extract information associated with a device node. |
| PARAMETERS | |
| All Interfaces | node A handle to a device node. |
| di_compatible_names() | names The address of a pointer. |
| RETURN VALUES | |
| di_binding_name() | di_binding_name() returns a pointer to the binding name. The binding name is the name used by the system to select a driver for the device. |
| di_bus_addr() | di_bus_addr() returns a pointer to a null-terminated string containing the assigned bus address for the device. NULL is returned if a bus address has not been assigned to the device. A zero-length string may be returned and is considered a valid bus address.\011 |
| di_compatible_names() | The return value of di_compatible_names() is the number of compatible names. <i>names</i> is updated to point to a buffer contained within the snapshot. The buffer contains a concatenation of null-terminated strings, for example: |

```
<
name1
>0<
name2
>0...<
namen
>0
```

See the discussion of generic names in *Writing Device Drivers* for a description of how compatible names are used by Solaris to achieve driver binding for the node.

- di_devid()** **di_devid()** returns the device ID for *node* , if it is registered. Otherwise, a null pointer is returned. Interfaces in the `libdevid(4)` library may be used to manipulate the handle to the device id. \011
- di_driver_name()** **di_driver_name()** returns the name of the driver bound to the *node* . A null pointer is returned if *node* is not bound to any driver. \011
- di_driver_ops()** **di_driver_ops()** returns a bit array of device driver entry points that are supported by the driver bound to this *node* . Possible bit fields supported by the driver are `DI_CB_OPS` , `DI_BUS_OPS` , `DI_STREAM_OPS` .
- di_instance()** **di_instance()** returns the instance number of the device. A value of -1 indicates an instance number has not been assigned to the device by the system.
- di_nodeid()** **di_nodeid()** returns the type of device, which may be one of the following possible values: `DI_PSEUDO_NODEID` , `DI_PROM_NODEID` , and `DI_SID_NODEID` . Devices of type `DI_PROM_NODEID` may have additional properties that are defined by the PROM . See `di_prom_prop_data(3)` and `di_prom_prop_lookup_bytes(3)` .
- di_node_name()** **di_node_name()** returns a pointer to a null-terminated string containing the node name.
- EXAMPLES** See `di_init(3)` for an example showing typical use of these interfaces.
- ATTRIBUTES** See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---------------------|-----------------|
| MT Level | Safe |
| Interface Stability | Evolving |

SEE ALSO

`di_init(3)`, `di_prom_init(3)`, `di_prom_prop_data(3)`,
`di_prom_prop_lookup_bytes(3)`, `libdevinfo(3)`, `libdevid(4)`,
`attributes(5)`

Writing Device Drivers

| | |
|--------------------|---|
| NAME | di_child_node, di_parent_node, di_sibling_node, di_drv_first_node, di_drv_next_node – libdevinfo node traversal functions |
| SYNOPSIS | <pre> di_node_t di_child_node(di_node_t node); di_node_t di_parent_node(di_node_t node); di_node_t di_sibling_node(di_node_t node); di_node_t di_drv_first_node(const char * drv_name, di_node_t root); di_node_t di_drv_next_node(di_node_t node); </pre> |
| DESCRIPTION | <p>The kernel device configuration data may be viewed in two ways, either as a tree of device configuration nodes or as a list of nodes associated with each driver. In the tree view, each node may contain references to its parent, the next sibling in a list of siblings, and the first child of a list of children. In the per-driver view, each node contains a reference to the next node associated with the same driver. \011\011</p> <p>Both views are captured in the snapshot, and the interfaces are provided for node access. \011</p> <p>di_child_node() obtains a handle to the first child of <i>node</i>. DI_NODE_NIL is returned and <i>errno</i> is set to ENXIO or ENOTSUP, if no child node exists in the snapshot.</p> <p>\011 di_parent_node() obtains a handle to the parent node of <i>node</i>. DI_NODE_NIL is returned and <i>errno</i> is set to ENXIO or ENOTSUP, if no parent node exists in the snapshot. \011</p> <p>di_sibling_node() obtains a handle to the next sibling node of <i>node</i>. A DI_NODE_NIL is returned and <i>errno</i> is set to ENXIO or ENOTSUP, if no next sibling node exists in the snapshot.\011</p> <p>di_drv_first_node() obtains a handle to the first node associated with the driver specified by <i>drv_name</i>. If there is no such driver, DI_NODE_NIL is returned with <i>errno</i> is set to EINVAL. If the driver exists, but there is no node associated with this driver, DI_NODE_NIL is returned and <i>errno</i> is set to ENXIO or ENOTSUP. \011</p> <p>di_drv_next_node() returns a handle to the next node bound to the same driver. DI_NODE_NIL is returned if no more nodes exist.</p> |
| PARAMETERS | <p>The following parameter descriptions apply to di_child_node(), di_drv_next_node(), di_parent_node(), and di_sibling_node() :</p> <p>node A handle to any node in the snapshot.</p> <p>The following parameter descriptions apply to di_drv_first_node() :</p> |

drv_name The name of the driver of interest.

root The handle of the root node for the snapshot returned by **di_init()**.

RETURN VALUES

Upon successful completion, a handle is returned. Otherwise, `DI_NODE_NIL` is returned and `errno` is set to indicate the error.

ERRORS

These functions set `errno` as listed for the following conditions:

EINVAL The argument is invalid.

ENXIO The requested node does not exist.

ENOTSUP The node was not found in the snapshot, but it may exist in the kernel. This error may occur if the snapshot contains a partial device tree.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---------------------|-----------------|
| MT Level | Safe |
| Interface Stability | Evolving |

SEE ALSO

libdevinfo(3), **attributes(5)**

Writing Device Drivers

NAME di_devfs_path, di_devfs_path_free – generate and free physical path names

SYNOPSIS

```
char * di_devfs_path(di_node_t node);
void di_devfs_path_free(char * path_buf);
```

DESCRIPTION

di_devfs_path() generates the physical path of the device *node* . The caller is responsible for freeing the memory allocated to store the physical path by calling **di_devfs_path_free()** .

di_devfs_path_free() frees memory that was allocated by **di_devfs_path()** .

PARAMETERS

di_devfs_path()

| | |
|-------------|--|
| <i>node</i> | Handle to a device node in the snapshot. |
|-------------|--|

di_devfs_path_free()

| | |
|-----------------|--|
| <i>path_buf</i> | Pointer returned by di_devfs_path() . |
|-----------------|--|

RETURN VALUES

di_devfs_path() Pointer to the string containing the physical path of *node* .

ERRORS

EINVAL *node* is not a valid handle.

di_devfs_path() also return any error code from **malloc(3C)** .

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---------------------|-----------------|
| MT Level | Safe |
| Interface Stability | Evolving |

SEE ALSO **malloc(3C)** , **libdevinfo(3)** , **attributes(5)**

Writing Device Drivers

NAME difftime – computes the difference between two calendar times

SYNOPSIS `#include <time.h>`
`double difftime(time_t time1, time_t time0);`

DESCRIPTION The **difftime()** function computes the difference between two calendar times.

RETURN VALUES The **difftime()** functions returns the difference (*time1-time0*) expressed in seconds as a double.

USAGE The **difftime()** function is provided because there are no general arithmetic properties defined for type `time_t`.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **ctime(3C)**, **attributes(5)**

| | |
|----------------------|--|
| NAME | di_init, di_fini – create and destroy a snapshot of kernel device tree |
| SYNOPSIS | <pre>di_node_t di_init(const char * phys_path, uint_t flags); void di_fini(di_node_t root);</pre> |
| DESCRIPTION | <p>di_init() creates a snapshot of the kernel device tree and returns a handle of the <i>root</i> node. The caller specifies the contents of the snapshot by providing <i>flag</i> and <i>phys_path</i> .</p> <p>di_fini() destroys the snapshot of the kernel device tree and frees the associated memory. All handles associated with this snapshot become invalid after the call to di_fini() .</p> |
| PARAMETERS | |
| di_init() | <p>phys_path Physical path of the <i>root</i> node of the snapshot. See di_devfs_path(3) .</p> <p>flags Snapshot content specification. The possible values may be a bitwise OR of the following:</p> <p style="margin-left: 40px;">DINFOSUBTREE Include subtree.</p> <p style="margin-left: 40px;">DINFOPROP Include properties.</p> <p style="margin-left: 40px;">DINFOMINOR Include minor data.</p> <p style="margin-left: 40px;">DINFOCOPYALL Include all of above.</p> <p>If <i>flags</i> is 0 , the snapshot contains only a single node without properties or minor nodes.</p> |
| di_fini() | <p>root Handle obtained by calling di_init() .</p> |
| RETURN VALUES | |
| di_init() | <p>Upon success, a handle is returned. Otherwise, DI_NODE_NIL is returned and <i>errno</i> is set to indicate the error.</p> |
| ERRORS | <p>di_init() may set <i>errno</i> to any error code that may also be set by open(2) , ioctl(2) or mmap(2) . The most common error codes include:</p> <p>EACCESS Insufficient privilege for accessing device configuration data.</p> |

- ENXIO** Either the device named by *phys_path* is not present in the system, or the `devinfo(7D)` driver is not installed properly.
- EINVAL** Either *phys_path* is incorrectly formed or the *flags* argument is invalid.

EXAMPLES**EXAMPLE 1** Using the `libdevinfo()` Interfaces To Print All Device Tree Node Names

The following is an example using the `libdevinfo()` interfaces to print all device tree node names:

```

/+
 * Code to print all device tree node names
 */

#include <stdio.h>
#include <libdevinfo.h>

int
prt_nodename(di_node_t node, void *arg)
{
    printf("%s\
", di_node_name(node));
    return (DI_WALK_CONTINUE);
}

main()
{
    di_node_t root_node;
    if((root_node = di_init("/", DINFOSUBTREE)) == DI_NODE_NIL) {
        fprintf(stderr, "di_init() failed\
");
        exit(1);
    }
    di_walk_node(root_node, DI_WALK_CLDFIRST, NULL, prt_nodename);
    di_fini(root_node);
}

```

EXAMPLE 2 Using the `libdevinfo()` Interfaces To Print The Physical Path Of SCSI Disks

The following example uses the `libdevinfo()` interfaces to print the physical path of SCSI disks:

```

/*
 * Code to print physical path of scsi disks
 */

```

```

#include <stdio.h>
#include <libdevinfo.h>
#define\011DISK_DRIVER\011"sd"\011/* driver name */

void
prt_diskinfo(di_node_t node)
{
    int instance;
\011    char *phys_path;

    /*
     * If the device node exports no minor nodes,
     * there is no physical disk.
     */
    if (di_minor_next(node, DI_MINOR_NIL) == DI_MINOR_NIL) {
\011\011    return;
\011    }

\011    instance = di_instance(node);
\011    phys_path = di_devfs_path(node);
\011    printf("%s%d: %s\
", DISK_DRIVER, instance, phys_path);
\011    di_devfs_path_free(phys_path);
}

void
walk_disknodes(di_node_t node)
{
\011    node = di_drv_first_node(DISK_DRIVER, node);
\011    while (node != DI_NODE_NIL) {
\011\011    prt_diskinfo(node);
\011\011    node = di_drv_next_node(node);
\011    }
}

main()
{
\011    di_node_t root_node;
    if ((root_node = di_init("/", DINFOCPYALL)) == DI_NODE_NIL) {
        fprintf(stderr, "di_init() failed\
");
        exit(1);
    }
\011    walk_disknodes(root_node);
\011    di_fini(root_node);
}

```

ATTRIBUTES

See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---------------------|-----------------|
| MT Level | Safe |
| Interface Stability | Evolving |

SEE ALSO `open(2)`, `ioctl(2)`, `mmap(2)`, `libdevinfo(3)`, `attributes(5)`

Writing Device Drivers

| NAME | di_minor_devt, di_minor_name, di_minor_nodetype, di_minor_spectype – return libdevinfo minor node information | | | | | | |
|----------------------------|---|----------------|-----------------|----------|------|---------------------|----------|
| SYNOPSIS | <pre>dev_t di_minor_devt(di_minor_t minor); char * di_minor_name(di_minor_t minor); char * di_minor_nodetype(di_minor_t minor); int di_minor_spectype(di_minor_t minor);</pre> | | | | | | |
| DESCRIPTION | These interfaces are used to return libdevinfo minor node information. | | | | | | |
| PARAMETERS | minor A handle to minor data node. | | | | | | |
| RETURN VALUES | | | | | | | |
| di_minor_name() | di_minor_name() \011returns the minor <i>name</i> . See ddi_create_minor_node(9F) for a description of the <i>name</i> parameter. | | | | | | |
| di_minor_devt() | The function di_minor_devt() returns the <code>dev_t</code> value of the minor node that is specified by SYS V ABI. See getmajor(9F) , getminor(9F) , and ddi_create_minor_node(9F) for more information. | | | | | | |
| di_minor_spectype() | di_minor_spectype() \011returns the <i>spec_type</i> of the file, either <code>S_IFCHR</code> or <code>S_IFBLK</code> . See\011 ddi_create_minor_node(9F) for a description of the <i>spec_type</i> parameter. | | | | | | |
| di_minor_nodetype() | di_minor_nodetype() returns the minor <i>node_type</i> of the minor node. See ddi_create_minor_node(9F) for a description of the <i>node_type</i> parameter. | | | | | | |
| ERRORS | No error codes are returned. | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT Level</td> <td>Safe</td> </tr> <tr> <td>Interface Stability</td> <td>Evolving</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT Level | Safe | Interface Stability | Evolving |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| MT Level | Safe | | | | | | |
| Interface Stability | Evolving | | | | | | |
| SEE ALSO | attributes(5) , ddi_create_minor_node(9F) , getmajor(9F) , getminor(9F) | | | | | | |

Writing Device Drivers

NAME di_minor_next – libdevinfo minor node traversal functions

SYNOPSIS di_minor_t di_minor_next(di_node_t node, di_minor_t minor);

DESCRIPTION **di_minor_next()** returns a handle to the next minor node for the device node *node*. If *minor* is DI_MINOR_NIL, a handle to the first minor node is returned.

PARAMETERS

node Device node with which the minor node is associated.

minor Handle to the current minor node or DI_MINOR_NIL.

RETURN VALUES Upon successful completion, a handle to the next minor node is returned. Otherwise, DI_MINOR_NIL is returned and *errno* is set to indicate the error.

ERRORS *errno* is set as listed for the following conditions:

EINVAL Invalid argument.

ENXIO End of minor node list.

ENOTSUP Minor node information is not available in snapshot.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---------------------|-----------------|
| MT Level | Safe |
| Interface Stability | Evolving |

SEE ALSO **libdevinfo(3)**, **attributes(5)**
Writing Device Drivers

NAME | `di_prom_init`, `di_prom_fini` – create and destroy a handle to the PROM device information

SYNOPSIS | `di_prom_handle_t di_prom_init();`
`void di_prom_fini(di_prom_handle_t ph);`

DESCRIPTION | For device nodes whose `nodeid` value is `DI_PROM_NODEID` (see `di_nodeid(3)`), additional properties may be retrieved from the PROM. `di_prom_init()` returns a handle that is used to retrieve such properties. This handle is passed to `di_prom_prop_lookup_bytes(3)` and `di_prom_prop_next(3)`. `di_prom_fini()` destroys the handle and all handles to PROM device information obtained from that handle.

PARAMETERS | *ph* Handle to `prom` returned by `di_prom_init()`.

RETURN VALUES

`di_prom_init()` | Upon successful completion, a handle is returned. Otherwise, `DI_PROM_HANDLE_NIL` is returned and `errno` is set to indicate the error.

ERRORS | `di_prom_init()` sets `errno` to any error code that may also be set by `openprom(7D)` or `malloc(3C)`.

ATTRIBUTES | See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---------------------|-----------------|
| MT Level | Safe |
| Interface Stability | Evolving |

SEE ALSO | `di_nodeid(3)`, `di_prom_prop_next(3)`, `di_prom_prop_lookup_bytes(3)`, `libdevinfo(3)`, `malloc(3C)`, `attributes(5)`, `openprom(7D)`

| | |
|----------------------|--|
| NAME | di_prom_prop_data, di_prom_prop_next, di_prom_prop_name – access PROM device information |
| SYNOPSIS | <pre>di_prom_prop_t di_prom_prop_next(di_prom_handle_t ph, di_node_t node, di_prom_prop_t prom_prop); char * di_prom_prop_name(di_prom_prop_t prom_prop); int di_prom_prop_data(di_prom_prop_t prom_prop, uchar_t ** prop_data);</pre> |
| DESCRIPTION | <p>di_prom_prop_next() obtains a handle to the next property on the PROM property list associated with <i>node</i> . If <i>prom_prop</i> is DI_PROM_PROP_NIL , the first property associated with <i>node</i> is returned.</p> <p>di_prom_prop_name() returns the name of the <i>prom_prop</i> property.</p> <p>di_prom_prop_data() returns the value of the <i>prom_prop</i> property. The return value is a non-negative integer specifying the size in number of bytes in <i>prop_data</i> .</p> <p>All memory allocated by these functions is managed by the library and must not be freed by the caller.</p> |
| PARAMETERS | |
| All Interfaces | |
| <i>prom_prop</i> | Handle to a PROM property. |
| di_prom_prop_data() | <i>prop_data</i> Address of a pointer. |
| di_prom_prop_next() | <i>ph</i> PROM handle |
| | <i>node</i> Handle to a device node in the snapshot of kernel device tree. |
| RETURN VALUES | |
| di_prom_prop_data() | di_prom_prop_data() returns the number of bytes in <i>prop_data</i> and <i>prop_data</i> is updated to point to a byte array containing the property value . If 0 is returned, the property is a boolean property, and the existence of this property indicates the value is true. |
| di_prom_prop_name() | di_prom_prop_name() returns a pointer to a string that contains the name of <i>prom_prop</i> . |

di_prom_prop_next()

di_prom_prop_next() returns a handle to the next PROM property. `DI_PROM_PROP_NIL` is returned if no additional properties exist.

ERRORS

See `openprom(7D)` for a description of possible errors.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---------------------|-----------------|
| MT Level | Safe |
| Interface Stability | Evolving |

SEE ALSO

`attributes(5)` , `openprom(7D)`

Writing Device Drivers

| | |
|----------------------|---|
| NAME | di_prom_prop_lookup_bytes, di_prom_prop_lookup_ints, di_prom_prop_lookup_strings – search for a PROM property |
| SYNOPSIS | <pre>int di_prom_prop_lookup_bytes(di_prom_handle_t ph, di_node_t node, const char * prop_name, uchar_t ** prop_data); int di_prom_prop_lookup_ints(di_prom_handle_t ph, di_node_t node, const char * prop_name, int ** prop_data); int di_prom_prop_lookup_strings(di_prom_handle_t ph, di_node_t node, const char * prop_name, char ** prop_data);</pre> |
| DESCRIPTION | These functions are used for returning the value of a known PROM property name and value type. These functions will update the <i>prop_data</i> pointer to reference memory that contains the property value. All memory allocated by these functions is managed by the library and must not be freed by the caller. |
| PARAMETERS | <p>The following parameter descriptions apply to all interfaces:</p> <p>node Handle to device node in snapshot created by <code>di_init(3)</code>.</p> <p>ph Handle returned by <code>di_prom_init(3)</code>.</p> <p>prop_name Name of the property being searched.</p> <p>The following parameter description applies to <code>di_prom_prop_lookup_bytes()</code> only:</p> <p>prop_data The address of a pointer to an array of unsigned characters.</p> <p>The following parameter description applies to <code>di_prom_prop_lookup_ints()</code> only:</p> <p>prop_data The address of a pointer to an integer.</p> <p>The following parameter description applies to <code>di_prom_prop_lookup_strings()</code> only:</p> <p>prop_data The address of pointer to a buffer.</p> |
| RETURN VALUES | <p>If the property is found, the number of entries in <i>prop_data</i> is returned. If the property is a boolean type, 0 is returned, and the existence of this property indicates the value is true. Otherwise, -1 is returned with <code>errno</code> set to indicate the error condition.</p> <p>For <code>di_prom_prop_lookup_bytes()</code>, the number of entries is the number of unsigned characters contained in the buffer pointed to by <i>prop_data</i>.</p> <p>For <code>di_prom_prop_lookup_ints()</code>, the number of entries is the number of integers contained in the buffer pointed to by <i>prop_data</i>.</p> |

For **di_prom_prop_lookup_strings()**, the number of entries is the number of null-terminated strings contained in the buffer. The strings are stored in a concatenated format in the buffer.

ERRORS

These functions set `errno` as listed for the following conditions:

EINVAL Invalid argument.

ENXIO The property does not exist.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---------------------|-----------------|
| MT Level | Safe |
| Interface Stability | Evolving |

SEE ALSO

di_init(3), **di_prom_prop_next(3)**, **libdevinfo(3)**, **attributes(5)**, **openprom(7D)**

Writing Device Drivers

| | | | | | | | | | | | |
|--------------------------|---|----------------------|---|------------------|--|---------------------|---|-------------------|---|--------------------------|---|
| NAME | di_prop_bytes, di_prop_devt, di_prop_ints, di_prop_name, di_prop_strings, di_prop_type - access property values and attributes | | | | | | | | | | |
| SYNOPSIS | <pre>int di_prop_bytes(di_prop_t prop, uchar_t ** prop_data); dev_t di_prop_devt(di_prop_t prop); int di_prop_ints(di_prop_t prop, int ** prop_data); char * di_prop_name(di_prop_t prop); int di_prop_strings(di_prop_t prop, char ** prop_data); int di_prop_type(di_prop_t prop);</pre> | | | | | | | | | | |
| DESCRIPTION | <p>These interfaces are used to access information associated with property values and attributes.</p> <p>All memory allocated by these functions is managed by the library and must not be freed by the caller.</p> <p>di_prop_name() returns the name of the property. \011</p> <p>di_prop_type() returns the type of the property. The type determines the appropriate interface to access property values. The following is a list of possible types:</p> <table border="0" style="width: 100%;"> <tr> <td style="vertical-align: top; padding-right: 20px;">DI_PROP_TYPE_BOOLEAN</td> <td>There is no interface to call since there is no property data associated with boolean properties. The existence of the property defines a TRUE value.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 20px;">DI_PROP_TYPE_INT</td> <td>Use di_prop_ints() to access property data.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 20px;">DI_PROP_TYPE_STRING</td> <td>Use di_prop_strings() to access property data.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 20px;">DI_PROP_TYPE_BYTE</td> <td>Use di_prop_bytes() to access property data.\011</td> </tr> <tr> <td style="vertical-align: top; padding-right: 20px;">DI_PROP_TYPE_UNKNOWN\011</td> <td>Use di_prop_bytes() to access property data.\011 Since the type of property is unknown, the caller is responsible for interpreting the contents of the data.</td> </tr> </table> | DI_PROP_TYPE_BOOLEAN | There is no interface to call since there is no property data associated with boolean properties. The existence of the property defines a TRUE value. | DI_PROP_TYPE_INT | Use di_prop_ints() to access property data. | DI_PROP_TYPE_STRING | Use di_prop_strings() to access property data. | DI_PROP_TYPE_BYTE | Use di_prop_bytes() to access property data.\011 | DI_PROP_TYPE_UNKNOWN\011 | Use di_prop_bytes() to access property data.\011 Since the type of property is unknown, the caller is responsible for interpreting the contents of the data. |
| DI_PROP_TYPE_BOOLEAN | There is no interface to call since there is no property data associated with boolean properties. The existence of the property defines a TRUE value. | | | | | | | | | | |
| DI_PROP_TYPE_INT | Use di_prop_ints() to access property data. | | | | | | | | | | |
| DI_PROP_TYPE_STRING | Use di_prop_strings() to access property data. | | | | | | | | | | |
| DI_PROP_TYPE_BYTE | Use di_prop_bytes() to access property data.\011 | | | | | | | | | | |
| DI_PROP_TYPE_UNKNOWN\011 | Use di_prop_bytes() to access property data.\011 Since the type of property is unknown, the caller is responsible for interpreting the contents of the data. | | | | | | | | | | |

DI_PROP_UNDEFINED

The property has been undefined by the driver. No property data is available.

di_prop_devt() returns the `dev_t` with which this property is associated. If the value is `DDI_DEV_T_NONE`, the property is not defined for a specific minor node.

di_prop_bytes() returns the property data as a series of unsigned characters.

di_prop_ints() returns the property data as a series of integers.

di_prop_strings() returns the property data as a concatenation of null-terminated strings.

PARAMETERS

All Interfaces

prop Handle to a property returned by `di_prop_next(3)`.

`di_prop_bytes`

prop_data The address of a pointer to an unsigned character.

`di_prop_ints`

prop_data The address of a pointer to an integer.

`di_prop_strings`

prop_data The address of pointer to a character.

RETURN VALUES

`di_prop_bytes`,
`di_prop_ints`,
`di_prop_strings`

Upon successful completion, these interfaces return a non-negative value, indicating the number of entries in the property value buffer. See `di_prop_lookup_bytes(3)` for a description of the return values. Otherwise, -1 is returned and `errno` is set to indicate the error condition.

`di_prop_devt`

di_prop_devt() returns the `dev_t` value associated with the property.

`di_prop_name`

di_prop_name() returns a pointer to a string containing the name of the property.

`di_prop_type`

di_prop_type() may return one of various types described in the DESCRIPTION section.

ERRORS

These functions set `errno` as listed for the following conditions:

EINVAL Invalid argument. For example, the property type does not match the interface.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---------------------|-----------------|
| MT Level | Safe |
| Interface Stability | Evolving |

SEE ALSO

di_prom_prop_lookup_bytes(3), **di_prop_next(3)**, **libdevinfo(3)**, **attributes(5)**

Writing Device Drivers

| | |
|-------------------------------|---|
| NAME | di_prop_lookup_bytes, di_prop_lookup_ints, di_prop_lookup_strings – search for a property |
| SYNOPSIS | <pre>int di_prop_lookup_bytes(dev_t dev, di_node_t node, const char * prop_name, uchar_t ** prop_data); int di_prop_lookup_ints(dev_t dev, di_node_t node, const char * prop_name, int ** prop_data); int di_prop_lookup_strings(dev_t dev, di_node_t node, const char * prop_name, char ** prop_data);</pre> |
| DESCRIPTION | <p>These functions are used for returning the value of a known property name type and <code>dev_t</code> value.</p> <p>All memory allocated by these functions is managed by the library and must not be freed by the caller.</p> |
| PARAMETERS | |
| All Interfaces | |
| | <p><i>dev</i> <code>dev_t</code> of minor node with which the property is associated. <code>DDI_DEV_T_ANY</code> is a wild card that matches all <code>dev_t</code> 's, including <code>DDI_DEV_T_NONE</code> .</p> <p><i>node</i> Handle to the device node with which the property is associated.</p> <p><i>prop_name</i> Name of the property for which to search.</p> |
| di_prop_lookup_bytes | <p><i>prop_data</i> Address to a pointer to an array of unsigned characters containing the property data.</p> |
| di_prop_lookup_ints | <p><i>prop_data</i> Address to a pointer to an array of integers containing the property data.</p> |
| di_prop_lookup_strings | <p><i>prop_data</i> Address to a pointer to a buffer containing a concatenation of null-terminated strings containing the property data.</p> |
| RETURN VALUES | <p>If the property is found, the number of entries in <i>prop_data</i> is returned. If the property is a boolean type, 0 is returned, and the existence of this property indicates the value is true. Otherwise, -1 is returned with <code>errno</code> set to indicate the error condition.</p> |

ERRORS

These functions set `errno` as listed for the following conditions:

- EINVAL** Invalid argument.
- ENOTSUP** The snapshot contains no property information.
- ENXIO** The property does not exist; try `di_prom_prop_lookup_*` .

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---------------------|-----------------|
| MT Level | Safe |
| Interface Stability | Evolving |

SEE ALSO

`di_init(3)` , `di_prom_prop_lookup_bytes(3)` , `libdevinfo(3)` , `attributes(5)`

Writing Device Drivers

| NAME | di_prop_next – libdevinfo property traversal function | | | | | | |
|----------------------|--|----------------|-----------------|----------|------|---------------------|----------|
| SYNOPSIS | di_prop_t di_prop_next(di_node_t node, di_prop_t prop); | | | | | | |
| DESCRIPTION | The function di_prop_next() returns a handle to the next property on the property list. If <i>prop</i> is DI_PROP_NIL, the handle to the first property is returned. | | | | | | |
| PARAMETERS | <p>node Handle to a device node.</p> <p>prop Handle to a property.</p> | | | | | | |
| RETURN VALUES | Upon successful completion, di_prop_next() returns a handle. Otherwise DI_PROP_NIL is returned, and <code>errno</code> is set to indicate the error condition. | | | | | | |
| ERRORS | <p>The di_prop_next() functions sets <code>errno</code> as listed for the following conditions:</p> <p>EINVAL Invalid argument.</p> <p>ENOTSUP Snapshot does not contain property information.</p> <p>ENXIO There are no more properties.</p> | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT Level</td> <td>Safe</td> </tr> <tr> <td>Interface Stability</td> <td>Evolving</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT Level | Safe | Interface Stability | Evolving |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| MT Level | Safe | | | | | | |
| Interface Stability | Evolving | | | | | | |
| SEE ALSO | <p>di_init(3), libdevinfo(3), attributes(5)</p> <p><i>Writing Device Drivers</i></p> | | | | | | |

| | |
|--------------------|--|
| NAME | directio – provide advice to file system |
| SYNOPSIS | <pre>#include <sys/types.h> #include <sys/fcntl.h> int directio(int <i>filde</i>, int <i>advice</i>);</pre> |
| DESCRIPTION | <p>The directio() function provides advice to the system about the expected behavior of the application when accessing the data in the file associated with the open file descriptor <i>filde</i>. The system uses this information to help optimize accesses to the file's data. The directio() function has no effect on the semantics of the other operations on the data, though it may affect the performance of other operations.</p> <p>The <i>advice</i> argument is kept per file; the last caller of directio() sets the <i>advice</i> for all applications using the file associated with <i>filde</i>.</p> <p>Values for <i>advice</i> are defined in <code><sys/fcntl.h></code>.</p> <p>DIRECTIO_OFF Applications get the default system behavior when accessing file data.</p> <p>When an application reads data from a file, the data is first cached in system memory and then copied into the application's buffer (see read(2)). If the system detects that the application is reading sequentially from a file, the system will asynchronously "read ahead" from the file into system memory so the data is immediately available for the next read(2) operation.</p> <p>When an application writes data into a file, the data is first cached in system memory and is written to the device at a later time (see write(2)). When possible, the system increases the performance of write(2) operations by cacheing the data in memory pages. The data is copied into system memory and the write(2) operation returns immediately to the application. The data is later written asynchronously to the device. When possible, the cached data is "clustered" into large chunks and written to the device in a single write operation.</p> <p>The system behavior for DIRECTIO_OFF can change without notice.</p> <p>DIRECTIO_ON The system behaves as though the application is not going to reuse the file data in the near future. In other words, the file data is not cached in the system's memory pages.</p> |

When possible, data is read or written directly between the application's memory and the device when the data is accessed with `read(2)` and `write(2)` operations. When such transfers are not possible, the system switches back to the default behavior, but just for that operation. In general, the transfer is possible when the application's buffer is aligned on a two-byte (short) boundary, the offset into the file is on a device sector boundary, and the size of the operation is a multiple of device sectors.

This advisory is ignored while the file associated with *fildev* is mapped (see `mmap(2)`).

The system behavior for `DIRECTIO_ON` can change without notice.

RETURN VALUES

Upon successful completion, `directio()` returns 0. Otherwise, it returns -1 and sets `errno` to indicate the error.

ERRORS

The `directio()` function will fail if:

- EBADF** The *fildev* argument is not a valid open file descriptor.
- ENOTTY** The *fildev* argument is not associated with a file system that accepts advisory functions.
- EINVAL** The value in *advice* is invalid.

USAGE

Small sequential I/O generally performs best with `DIRECTIO_OFF`.

Large sequential I/O generally performs best with `DIRECTIO_ON`, except when a file is sparse or is being extended and is opened with `O_SYNC` or `O_DSYNC` (see `open(2)`).

The `directio()` function is supported for the ufs file system type (see `fstyp(1M)`).

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`fstyp(1M)`, `mmap(2)`, `open(2)`, `read(2)`, `write(2)`, `attributes(5)`, `fcntl(5)`

WARNINGS

Switching between `DIRECTIO_OFF` and `DIRECTIO_ON` can slow the system because each switch to `DIRECTIO_ON` may entail flushing the file's data from the system's memory.

| NAME | dirname – report the parent directory name of a file path name | | | | | | | | | | | | | | |
|----------------------|---|--------------|---------------|------------|--------|---------|-----|-------|-----|-----|-----|----|----|------|----|
| SYNOPSIS | <pre>#include <libgen.h> char *dirname(char *path);</pre> | | | | | | | | | | | | | | |
| DESCRIPTION | <p>The dirname() function takes a pointer to a character string that contains a pathname, and returns a pointer to a string that is a pathname of the parent directory of that file. Trailing '/' characters in the path are not counted as part of the path.</p> <p>If <i>path</i> does not contain a '/', then dirname() returns a pointer to the string ".". If <i>path</i> is a null pointer or points to an empty string, dirname() returns a pointer to the string ".".</p> | | | | | | | | | | | | | | |
| RETURN VALUES | The dirname() function returns a pointer to a string that is the parent directory of <i>path</i> . If <i>path</i> is a null pointer or points to an empty string, a pointer to a string "." is returned. | | | | | | | | | | | | | | |
| ERRORS | No errors are defined. | | | | | | | | | | | | | | |
| EXAMPLES | <p>EXAMPLE 1 A sample code using the dirname() function.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Input String</th> <th style="text-align: center;">Output String</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">"/usr/lib"</td> <td style="text-align: center;">"/usr"</td> </tr> <tr> <td style="text-align: center;">"/usr/"</td> <td style="text-align: center;">"/"</td> </tr> <tr> <td style="text-align: center;">"usr"</td> <td style="text-align: center;">"/"</td> </tr> <tr> <td style="text-align: center;">"/"</td> <td style="text-align: center;">"/"</td> </tr> <tr> <td style="text-align: center;">."</td> <td style="text-align: center;">."</td> </tr> <tr> <td style="text-align: center;">".."</td> <td style="text-align: center;">."</td> </tr> </tbody> </table> <p>The following code fragment reads a path name, changes directory to the parent directory of the named file (see chdir(2)), and opens the file.</p> <pre>char path[100], *pathcopy; int fd; gets (path); pathcopy = strdup (path); chdir (dirname (pathcopy)); free (pathcopy); fd = open (basename (path), O_RDONLY);</pre> | Input String | Output String | "/usr/lib" | "/usr" | "/usr/" | "/" | "usr" | "/" | "/" | "/" | ." | ." | ".." | ." |
| Input String | Output String | | | | | | | | | | | | | | |
| "/usr/lib" | "/usr" | | | | | | | | | | | | | | |
| "/usr/" | "/" | | | | | | | | | | | | | | |
| "usr" | "/" | | | | | | | | | | | | | | |
| "/" | "/" | | | | | | | | | | | | | | |
| ." | ." | | | | | | | | | | | | | | |
| ".." | ." | | | | | | | | | | | | | | |
| USAGE | The dirname() function may modify the string pointed to by <i>path</i> , and may return a pointer to static storage that may then be overwritten by subsequent calls to dirname() . | | | | | | | | | | | | | | |

The **dirname()** and **basename(3C)** functions together yield a complete pathname. The expression `dirname(path)` obtains the pathname of the directory where `basename(path)` is found.

When compiling multithreaded applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multithreaded applications.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

basename(1), **chdir(2)**, **basename(3C)**, **attributes(5)**

NAME | DisconnectToServer – disconnect from a DMI service provider

SYNOPSIS |

```
cc [ flag ... ] file ... -ldmici -ldmimi [ library ... ]
#include <dmi/api.hh>

bool_t DisconnectToServer(DmiRpcHandle *dmi_rpc_handle);
```

DESCRIPTION | The **DisconnectToServer()** function disconnects a management application or a component instrumentation from a DMI service provider.

RETURN VALUES | The **ConnectToServer()** function returns TRUE if successful, otherwise FALSE.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-level | Safe |

SEE ALSO | **ConnectToServer(3X)**, **attributes(5)**

| NAME | div, ldiv, lldiv – compute the quotient and remainder | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <stdlib.h> div_t div(int numer, int denom); ldiv_t ldiv(long int numer, long int denom); lldiv_t lldiv(long long numer, long long denom);</pre> | | | | |
| DESCRIPTION | <p>The div() function computes the quotient and remainder of the division of the numerator <i>numer</i> by the denominator <i>denom</i> . It provides a well-defined semantics for the signed integral division and remainder operations, unlike the implementation-defined semantics of the built-in operations. The sign of the resulting quotient is that of the algebraic quotient, and if the division is inexact, the magnitude of the resulting quotient is the largest integer less than the magnitude of the algebraic quotient. If the result cannot be represented, the behavior is undefined; otherwise, <i>quotient</i> * <i>denom</i> + <i>remainder</i> will equal <i>numer</i> .</p> <p>The ldiv() and lldiv() functions are similar to div() , except that the arguments and the members of the returned structure are different. The ldiv() function returns a structure of type <code>ldiv_t</code> and has type <code>long int</code> . The lldiv() function returns a structure of type <code>lldiv_t</code> and has type <code>long long</code> .</p> | | | | |
| RETURN VALUES | <p>The div() function returns a structure of type <code>div_t</code> , comprising both the quotient and remainder:</p> <pre>int quot; /*quotient*/ int rem; /*remainder*/</pre> <p>The ldiv() function returns a structure of type <code>ldiv_t</code> and lldiv() returns a structure of type <code>lldiv_t</code> , comprising both the quotient and remainder:</p> <pre>long int quot; /*quotient*/ long int rem; /*remainder*/</pre> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | attributes(5) | | | | |

| | |
|-----------------------|---|
| NAME | di_walk_minor – traverse libdevinfo minor nodes |
| SYNOPSIS | int di_walk_minor (di_node_t <i>root</i> , const char * <i>minor_nodetype</i> , uint_t <i>flag</i> , void * <i>arg</i> , int (* <i>minor_callback</i>)(di_node_t <i>node</i> , di_minor_t <i>minor</i> , void * <i>arg</i>)); |
| DESCRIPTION | di_walk_minor () visits all minor nodes attached to device nodes in a subtree rooted at <i>root</i> . For each minor node that matches <i>minor_nodetype</i> , the caller-supplied function minor_callback () is invoked. The walk terminates immediately when minor_callback () returns DI_WALK_TERMINATE . |
| PARAMETERS | |
| di_walk_minor | <p>root Root of subtree to visit.</p> <p>minor_nodetype A character string specifying the minor data type, which may be one of the types defined by the Solaris DDI framework, for example, DDI_NT_BLOCK. NULL matches all <i>minor_node</i> types. See ddi_create_minor_node(9F).</p> <p>flag Specify 0. Reserved for future use.</p> <p>arg Pointer to caller- specific user data.</p> |
| minor_callback | <p>node The device node with which the minor node is associated.</p> <p>minor The minor node visited.</p> <p>arg Pointer to caller-specific data.</p> |
| RETURN VALUES | |
| di_walk_minor | Upon successful completion, 0 is returned. Otherwise, -1 is returned and errno is set to indicate the error. |
| minor_callback | <p>The allowed return values are:</p> <p>DI_WALK_CONTINUE Continue to visit subsequent minor data nodes.</p> <p>DI_WALK_TERMINATE Terminate the walk immediately.</p> |
| ERRORS | |
| di_walk_minor | EINVAL Invalid argument. |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---------------------|-----------------|
| MT Level | Safe |
| Interface Stability | Evolving |

SEE ALSO

di_minor_nodetype(3), **libdevinfo(3)**, **attributes(5)**
ddi_create_minor_node(9F)

Writing Device Drivers

| | |
|----------------------|---|
| NAME | di_walk_node – traverse libdevinfo device nodes |
| SYNOPSIS | int di_walk_node(di_node_t root, uint_t flag, void *arg, int (*node_callback)(di_node_t node, void *arg)); |
| DESCRIPTION | di_walk_node() visits all nodes in the subtree rooted at <i>root</i> . For each node found, the caller-supplied function node_callback() is invoked. The return value of node_callback() specifies subsequent walking behavior. |
| PARAMETERS | |
| di_walk_node | <p>root Handle to the root node of the subtree to visit.</p> <p>flag Specifies walking order, either <code>DI_WALK_CLDFIRST</code> (depth first) or <code>DI_WALK_SIBFIRST</code> (breadth first). <code>DI_WALK_CLDFIRST</code> is the default.</p> <p>arg Pointer to caller-specific data.</p> |
| node_callback | <p>node The node being visited.</p> <p>arg Pointer to caller-specific data.</p> |
| RETURN VALUES | |
| di_walk_node | 0 is returned upon success. Otherwise, -1 is returned, and <code>errno</code> is set to indicate the error. |
| node_callback | <p>The allowed return values are:</p> <p><code>DI_WALK_CONTINUE</code> Continue walking.</p> <p><code>DI_WALK_PRUNESIB</code> Continue walking, but skip siblings and their child nodes.</p> <p><code>DI_WALK_PRUNECHILD</code> Continue walking, but skip subtree rooted at current node .</p> <p><code>DI_WALK_TERMINATE</code> Terminate the walk immediately.</p> |
| ERRORS | |
| di_walk_node | EINVAL Invalid argument. |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---------------------|-----------------|
| MT Level | Safe |
| Interface Stability | Evolving |

SEE ALSO

di_init(3), **libdevinfo(3)**, **attributes(5)**

Writing Device Drivers

| | |
|----------------------|---|
| NAME | dladdr – translate address to symbolic information |
| SYNOPSIS | <pre>cc [flag ...] file... -ldl [library ...] #include <dlfcn.h> int dladdr(void *address, Dl_info *dlip);</pre> |
| DESCRIPTION | <p>dladdr() is one of a family of routines that give the user direct access to the dynamic linking facilities. (See <i>Linker and Libraries Guide</i>). These routines are made available via the library loaded when the option <code>-ldl</code> is passed to the link-editor.</p> <p>Note: <i>These routines are available to dynamically-linked processes ONLY.</i></p> <p>dladdr() determines if the specified <i>address</i> is located within one of the mapped objects that make up the current applications address space. An address is deemed to fall within a mapped object when it is between the base address, and the <code>_end</code> address of that object. If a mapped object fits this criteria, the symbol table made available to the run-time linker is searched to locate the nearest symbol to the specified address. The nearest symbol is one that has a value less than or equal to the required address.</p> <p>The <code>Dl_info</code> structure must be preallocated by the user. The structure members are filled in by dladdr() based on the specified <i>address</i>. The <code>Dl_info</code> structure includes the following members:</p> <pre>const char * dli_fname; void * dli_fbase; const char * dli_sname; void * dli_saddr;</pre> <p>Descriptions of these members appear below.</p> <p><code>dli_fname</code> Contains a pointer to the filename of the containing object.</p> <p><code>dli_fbase</code> Contains the base address of the containing object.</p> <p><code>dli_sname</code> Contains a pointer to the symbol name nearest to the specified address. This symbol either has the same address, or is the nearest symbol with a lower address.</p> <p><code>dli_saddr</code> Contains the actual address of the above symbol.</p> |
| RETURN VALUES | <p>If the specified <i>address</i> cannot be matched to a mapped object, a 0 is returned. Otherwise, a non-zero return is made and the associated <code>Dl_info</code> elements are filled.</p> |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

ld(1), **dlclose(3X)**, **dldump(3X)**, **dlerror(3X)**, **dlopen(3X)**, **dlsym(3X)**, **attributes(5)**

Linker and Libraries Guide

NOTES

The `Dl_info` pointer elements point to addresses within the mapped objects. These may become invalid if objects are removed prior to these elements being used (see **dlclose()**).

If no symbol is found to describe the specified address, both the `dli_sname` and `dli_saddr` members are set to 0.

| NAME | dlclose – close a shared object | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag ...] file ... -ldl [library ...] #include <dlfcn.h> int dlclose(void *handle);</pre> | | | | |
| DESCRIPTION | <p>dlclose() is one of a family of routines that give the user direct access to the dynamic linking facilities. (See <i>Linker and Libraries Guide</i>). These routines are made available via the library loaded when the option <code>-ldl</code> is passed to the link-editor.</p> <p>Note: <i>These routines are available to dynamically-linked processes ONLY.</i></p> <p>dlclose() disassociates a shared object previously opened by dlopen() from the current process. Once an object has been closed using dlclose(), its symbols are no longer available to dlsym(). All objects loaded automatically as a result of invoking dlopen() on the referenced object are also closed. <i>handle</i> is the value returned by a previous invocation of dlopen().</p> | | | | |
| RETURN VALUES | <p>If the referenced object was successfully closed, dlclose() returns 0. If the object could not be closed, or if <i>handle</i> does not refer to an open object, dlclose() returns a non-zero value. More detailed diagnostic information will be available through dlderror().</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | <p>ld(1), dladdr(3X), dldump(3X), dlderror(3X), dlopen(3X), dlsym(3X), attributes(5)</p> <p><i>Linker and Libraries Guide</i></p> | | | | |
| NOTES | <p>A successful invocation of dlclose() does not guarantee that the objects associated with <i>handle</i> will actually be removed from the address space of the process. Objects loaded by one invocation of dlopen() may also be loaded by another invocation of dlopen(). The same object may also be opened multiple times. An object will not be removed from the address space until all references to that object through an explicit dlopen() invocation have been closed and all other objects implicitly referencing that object have also been closed.</p> <p>Once an object has been closed by dlclose(), referencing symbols contained in that object can cause undefined behavior.</p> | | | | |

| | |
|--------------------|---|
| NAME | dldump – create a new file from a dynamic object component of the calling process |
| SYNOPSIS | <pre>cc [flag ...] file ... -ldl [library ...] #include <dlfcn.h> int dldump(const char * ipath, const char * opath, int flags);</pre> |
| DESCRIPTION | <p>dldump() is one of a family of routines that give the user direct access to the dynamic linking facilities. (See <i>Linker and Libraries Guide</i>). These routines are made available via the library loaded when the option <code>-ldl</code> is passed to the link-editor.</p> <p>Note: <i>These routines are available to dynamically-linked processes ONLY.</i></p> <p>dldump() creates a new dynamic object <i>opath</i> from an existing dynamic object <i>ipath</i> that is bound to the current process. An <i>ipath</i> value of 0 is interpreted as the dynamic object that started the process. The new object is constructed from the existing objects' disc file. Relocations can be applied to the new object to pre-bind it to other dynamic objects, or fix the object to a specific memory location. In addition, data elements within the new object may be obtained from the objects' memory image as it exists in the calling process.</p> <p>These techniques allow the new object to be executed with a lower startup cost, either because there are less relocations required to load the object, or because of a reduction in the data processing requirements of the object. However, it is important to note that limitations may exist in using these techniques. Applying relocations to the new dynamic object <i>opath</i> may restrict its flexibility within a dynamically changing environment. In addition, limitations regarding data usage may make dumping a memory image impractical (see <code>EXAMPLES</code>).</p> <p>The runtime linker verifies that the dynamic object <i>ipath</i> is mapped as part of the current process. Thus, the object must either be the dynamic object that started the process (see <code>exec(2)</code>), one of the process's dependencies, or an object that has been preloaded (see <code>ld.so.1(1)</code>).</p> <p>As part of the runtime processing of a dynamic object, <i>relocation</i> records within the object are interpreted and applied to offsets within the object. These offsets are said to be <i>relocated</i>. Relocations can be categorized into two basic types: <i>non-symbolic</i> and <i>symbolic</i>.</p> <p>The <i>non-symbolic</i> relocation is a simple <i>relative</i> relocation that requires the base address at which the object is mapped to perform the relocation. The <i>symbolic</i> relocation requires the address of an associated symbol, and results in a <i>binding</i> to the dynamic object that defines this symbol. This symbol definition may originate from any of the dynamic objects that make up the process, that is, the object that started the process, one of the process's dependencies, an object that has been preloaded, or the dynamic object being relocated.</p> |

The *flags* parameter controls the relocation processing and other attributes of producing the new dynamic object *opath*. Without any *flags*, the new object is constructed solely from the contents of the *ipath* disc file without any relocations applied.

Various relocation flags may be OR'ed into the *flags* parameter to affect the relocations applied to the new object. *Non-symbolic* relocations can be applied using the following:

RTLD_REL_RELATIVE Relocation records from the object *ipath*, that define *relative* relocations, are applied to the object *opath*.

A variety of *symbolic* relocations can be applied using the following flags (each of these flags also implies RTLD_REL_RELATIVE is in effect):

RTLD_REL_EXEC Symbolic relocations that result in binding *ipath* to the dynamic object that started the process (commonly a dynamic executable) are applied to the object *opath*.

RTLD_REL_DEPENDS Symbolic relocations that result in binding *ipath* to any of the dynamic dependencies of the process are applied to the object *opath*.

RTLD_REL_PRELOAD Symbolic relocations that result in binding *ipath* to any objects preloaded with the process are applied to the object *opath*. (See LD_PRELOAD in *ld.so.1(1)*).

RTLD_REL_SELF Symbolic relocations that result in binding *ipath* to itself are applied to the object *opath*.

RTLD_REL_ALL All relocation records defined in the object *ipath* are applied to the new object *opath* (this is basically a concatenation of all the above relocation flags).

Note that for dynamic executables, RTLD_REL_RELATIVE, RTLD_REL_EXEC, and RTLD_REL_SELF have no effect (see EXAMPLES).

If relocations, knowledgeable of the base address of the mapped object, are applied to the new object *opath*, then the new object will become fixed to the location that the *ipath* image is mapped within the current process.

Any relocations applied to the new object *opath* will have the original relocation record removed so that the relocation will not be applied more than once. Otherwise, the new object *opath* will retain the relocation records as they exist in the *ipath* disc file.

The following additional attributes for creating the new dynamic object *opath* can be specified using the *flags* parameter:

| | |
|-------------|--|
| RTLD_MEMORY | The new object <i>opath</i> is constructed from the current memory contents of the <i>ipath</i> image as it exists in the calling process. This option allows data modified by the calling process to be captured in the new object. Note that not all data modifications may be applicable for capture; significant restrictions exist in using this technique (see <code>EXAMPLES</code>). By default, when processing a dynamic executable, any allocated memory that follows the end of the data segment is captured in the new object (see <code>malloc(3C)</code> and <code>brk(2)</code>). This data, which represents the process heap, is saved as a new <code>.SUNW_heap</code> section in the object <i>opath</i> . The objects' program headers and symbol entries, such as <code>_end</code> , are adjusted accordingly. See also <code>RTLD_NOHEAP</code> . When using this attribute, any relocations that have been applied to the <i>ipath</i> memory image that do not fall into one of the requested relocation categories are undone, that is, the relocated element is returned to the value as it existed in the <i>ipath</i> disc file. |
| RTLD_STRIP | Only collect allocatable sections within the object <i>opath</i> ; sections that are not part of the dynamic objects' memory image are removed. This parameter reduces the size of the <i>opath</i> disc file and is comparable to having run the new object through <code>strip(1)</code> . |
| RTLD_NOHEAP | Do not save any heap to the new object. This option is only meaningful when processing a dynamic executable with the <code>RTLD_MEMORY</code> attribute and allows for reducing the size of the <i>opath</i> disc file. In this case, the executable must confine its data initialization to data elements within its data segment and must not use any allocated data elements that comprise the heap. |

It should be emphasized that an object created by `dldump()` is simply an updated ELF object file. No additional state regarding the process at the time `dldump()` is called is maintained in the new object. `dldump()` does not provide a panacea for checkpoint/resume. A new dynamic executable, for

RETURN VALUES

example, will not start where the original executable called **dldump()**; it will gain control at the executable's normal entry point (see **EXAMPLES**).

On successful creation of the new object, **dldump()** returns 0. Otherwise, a non-zero value is returned and more detailed diagnostic information is available through **dlderror()**.

EXAMPLES

EXAMPLE 1 Sample code using **dldump()**.

The following code fragment, which can be part of a dynamic executable `a.out`, can be used to create a new shared object from one of the dynamic executables' dependencies `libfoo.so.1`:

```
const char * ipath = "libfoo.so.1";
const char * opath = "./tmp/libfoo.so.1";
...
if (dldump(ipath, opath, RTLD_REL_RELATIVE) != 0)
    (void) printf("dldump failed: %s\n", dlderror());
```

The new shared object *opath* is fixed to the address of the mapped *ipath* bound to the dynamic executable `a.out`. All relative relocations are applied to this new shared object, which will reduce its relocation overhead when it is used as part of another process.

By performing only relative relocations, any symbolic relocation records remain defined within the new object, and thus the dynamic binding to external symbols will be preserved when the new object is used.

Use of the other relocation flags can fix specific relocations in the new object and thus can reduce even more the runtime relocation startup cost of the new object. However, this will also restrict the flexibility of using the new object within a dynamically changing environment, as it will bind the new object to some or all of the dynamic objects presently mapped as part of the process.

For example, the use of `RTLD_REL_SELF` will cause any references to symbols from *ipath* to be bound to definitions within itself if no other preceding object defined the same symbol. In other words, a call to *foo()* within *ipath* will bind to the definition *foo* within the same object. Therefore, *opath* will have one less binding that must be computed at runtime. This reduces the startup cost of using *opath* by other applications; however, interposition of the symbol *foo* will no longer be possible.

Using a dumped shared object with applied relocations as an applications dependency normally requires that the application have the same dependencies as the application that produced the dumped image. Dumping shared objects, and the various flags associated with relocation processing, have some specialized uses. However, the technique is intended as a building block for future technology.

The following code fragment, which is part of the dynamic executable `a.out`, can be used to create a new version of the dynamic executable:

```
static char *      dumped = 0;
const char *      opath = "./a.out.new";
...
if (dumped == 0) {
    char          buffer[100];
    int           size;
    time_t        seconds;
    ...
    /* Perform data initialization */
    seconds = time((time_t *)0);
    size = cftime(buffer, (char *)0, &seconds);
    if ((dumped = (char *)malloc(size + 1)) == 0) {
        (void) printf("malloc failed: %s\n", strerror(errno));
        return (1);
    }
    (void) strcpy(dumped, buffer);
    ...
    /*
     * Tear down any undesirable data initializations and
     * dump the dynamic executables memory image.
     */
    _exithandle();
    _exit(dldump(0, opath, RTLD_MEMORY));
}
(void) printf("Dumped: %s\n", dumped);
```

Any modifications made to the dynamic executable, up to the point the `dldump()` call is made, are saved in the new object `a.out.new`. This mechanism allows the executable to update parts of its data segment and heap prior to creating the new object. In this case, the date the executable is dumped is saved in the new object. The new object can then be executed without having to carry out the same (presumably expensive) initialization.

For greatest flexibility, this example does not save *any* relocated information. The elements of the dynamic executable *ipath* that have been modified by relocations at process startup, that is, references to external functions, are returned to the values of these elements as they existed in the *ipath* disc file. This preservation of relocation records allows the new dynamic executable to be flexible, and correctly bind and initialize to its dependencies when executed on the same or newer upgrades of the OS.

Fixing relocations by applying some of the relocation flags would bind the new object to the dependencies presently mapped as part of the process calling `dldump()`. It may also remove necessary copy relocation processing required for the correct initialization of its shared object dependencies. Therefore, if the new dynamic executables' dependencies have no specialized initialization requirements, the executable may still only interact correctly with the

dependencies to which it binds if they were mapped to the same locations as they were when **dldump()** was called.

Note that for dynamic executables, `RTLD_REL_RELATIVE`, `RTLD_REL_EXEC`, and `RTLD_REL_SELF` have no effect, as relocations within the dynamic executable will have been fixed when it was created by `ld(1)`.

When `RTLD_MEMORY` is used, care should be taken to insure that dumped data sections that reference external objects are not reused without appropriate re-initialization. For example, if a data item contains a file descriptor, a variable returned from a shared object, or some other external data, and this data item has been initialized prior to the **dldump()** call, its value will have no meaning in the new dumped image.

When `RTLD_MEMORY` is used, any modification to a data item that is initialized via a relocation whose relocation record will be retained in the new image will effectively be lost or invalidated within the new image. For example, if a pointer to an external object is incremented prior to the **dldump()** call, this data item will be reset to its disc file contents so that it can be relocated when the new image is used; hence, the previous increment is lost.

Non-idempotent data initializations may prevent the use of `RTLD_MEMORY`. For example, the addition of elements to a linked-list via `init` sections can result in the linked-list data being captured in the new image. Running this new image may result in `init` sections continuing to add new elements to the list without the prerequisite initialization of the list head. It is recommended that `_exithandle(3C)` be called before **dldump()** to tear down any data initializations established via initialization code. Note that this may invalidate the calling image; thus, following the call to **dldump()**, only a call to `_exit(2)` should be made.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| Availability | SUNWcsu |
| MT-Level | MT-Safe |

SEE ALSO

`ld(1)`, `ld.so.1(1)`, `strip(1)`, `_exit(2)`, `brk(2)`, `exec(2)`, `_exithandle(3C)`, `dladdr(3X)`, `dlclose(3X)`, `dLError(3X)`, `dlopen(3X)`, `dlsym(3X)`, `end(3C)`, `malloc(3C)`, `attributes(5)`

Linker and Libraries Guide

NOTES

Any `NOBITS` sections within the *ipath* are expanded to `PROGBITS` sections within the *opath*. `NOBITS` sections occupy no space within an ELF file image.

They declare memory that must be created and zero-filled when the object is mapped into the runtime environment. *.bss* is a typical example of this section type. *PROGBITS* sections, on the other hand, hold information defined by the object within the ELF file image. This section conversion reduces the runtime initialization cost of the new dumped object but increases the objects' disc space requirement.

When a shared object is dumped, and relocations are applied which are knowledgeable of the base address of the mapped object, the new object is fixed to this new base address and thus its ELF type is reclassified to be a dynamic executable. This new object can be processed by the runtime linker, but is not valid as input to the link-editor.

If relocations are applied to the new object, any remaining relocation records will be reorganized for better locality of reference. The relocation sections are renamed to *.SUNW_reloc* and the association to the section they were to relocate is lost. Only the offset of the relocation record itself is meaningful. This change does not make the new object invalid to either the runtime linker or link-editor, but may reduce the objects analysis with some ELF readers.

| NAME | dlerror – get diagnostic information | | | | |
|--------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag ...] file ... -ldl [library ...] #include <dlfcn.h> char *dlerror(void);</pre> | | | | |
| DESCRIPTION | <p>dlerror() is one of a family of routines that give the user direct access to the dynamic linking facilities. (See <i>Linker and Libraries Guide</i>). These routines are made available via the library loaded when the option <code>-ldl</code> is passed to the link-editor.</p> <p>Note: <i>These routines are available to dynamically-linked processes ONLY.</i></p> <p>dlerror() returns a null-terminated character string (with no trailing newline) that describes the last error that occurred during dynamic linking processing. If no dynamic linking errors have occurred since the last invocation of dlerror(), dlerror() returns <code>NULL</code>. Thus, invoking dlerror() a second time, immediately following a prior invocation, will result in <code>NULL</code> being returned.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | <p>ld(1), dladdr(3X), dlclose(3X), dldump(3X), dlopen(3X), dlsym(3X), attributes(5)</p> <p><i>Linker and Libraries Guide</i></p> | | | | |
| NOTES | <p>The messages returned by dlerror() may reside in a static buffer that is overwritten on each call to dlerror(). Application code should not write to this buffer. Programs wishing to preserve an error message should make their own copies of that message.</p> | | | | |

| | |
|--------------------|--|
| NAME | dldlfo – dynamic load information |
| SYNOPSIS | <pre>cc [flag ...] file ... -ldl [library ...] #include <dldlfn.h> int dldlfo(void *handle, int request, void *p);</pre> |
| DESCRIPTION | <p>dldlfo() extracts information about a dynamically-loaded object. This interface is loosely modeled after the ioctl() interface. <i>request</i> and a third argument with varying type are passed to dldlfo(). The action taken by dldlfo() depends on the value of the <i>request</i> provided. <i>handle</i> is a value returned from a dldlopen() or dldmopen() call.</p> <p>The following are possible values for <i>request</i> to be passed into dldlfo():</p> <p>RTLD_DI_LMID obtains the id for the link-map list upon which the <i>handle</i> is loaded. <i>p</i> is a <code>Lmid_t</code> pointer (<code>Lmid_t *p</code>).</p> <p>RTLD_DI_LINKMAP obtains the <code>Link_map</code> for the <i>handle</i> specified. <i>p</i> points to a <code>Link_map</code> pointer (<code>Link_map **p</code>). The actual storage for the <code>Link_map</code> structure is maintained by <code>ld.so.l</code>.</p> <p>The <code>Link_map</code> structure includes the following members:</p> <pre>unsigned long l_addr; /* base address */ char * l_name; /* object name */ Elf32_Dyn * l_ld; /* .dynamic section */ Link_map * l_next; /* next link object */ Link_map * l_prev; /* previous link object */ char * l_refname; /* filter reference name */</pre> <p><code>l_addr</code> The base address of the object loaded into memory.</p> <p><code>l_name</code> Full name of the loaded object. This is the filename of the object as referenced by <code>ld.so.l</code>.</p> <p><code>l_ld</code> Points to the <code>SHT_DYNAMIC</code> structure.</p> <p><code>l_next</code> The next <code>Link_map</code> on the link-map list, other objects on the same link-map list as the current object may be examined by following the and <code>l_prev</code> fields.</p> <p><code>l_prev</code> The previous <code>Link_map</code> on the link-map list.</p> <p><code>l_refname</code> If the object referenced is a <i>filter</i> this field points to the name of the object being filtered. If the object is not a <i>filter</i>, this field will be 0. See <i>Linker and Libraries Guide</i></p> |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | MT-Safe |

SEE ALSO

`ld(1)`, `ioctl(2)`, `dlclose(3X)`, `dldump(3X)`, `dlerror(3X)`, `dlmopen(3X)`, `dlopen(3X)`, `dlsym(3X)`, `attributes(5)`

Linker and Libraries Guide

NOTES

These routines are available to dynamically-linked processes only.

| | |
|--------------------|--|
| NAME | dlopen, dlmopen – gain access to an executable object file |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -l dl [<i>library</i> ...] #include <dlfcn.h> void * dlopen(const char * <i>pathname</i>, int <i>mode</i>); void * dlmopen(Lmid_t <i>lmid</i>, const char * <i>pathname</i>, int <i>mode</i>);</pre> |
| DESCRIPTION | <p>dlopen() and dlmopen() are members of a family of routines that give the user direct access to the dynamic linking facilities. (See <i>Linker and Libraries Guide</i>). These routines are made available through the library loaded when the option <code>-l dl</code> is passed to the link-editor.</p> <p>Note: <i>These routines are available to dynamically-linked processes ONLY.</i></p> <p>dlopen() makes an executable object file available to a running process. dlopen() returns to the process a <i>handle</i> which the process may use on subsequent calls to dlsym() and dlclose(). The value of this <i>handle</i> should not be interpreted in any way by the process. <i>pathname</i> is the path name of the object to be opened. A path name containing an embedded <code>' / '</code> is interpreted as an absolute path or relative to the current directory; otherwise, the set of search paths currently in effect by the runtime linker will be used to locate the specified file. See NOTES below.</p> <p>Any dependencies recorded within <i>pathname</i> are also loaded as part of the dlopen(). These dependencies are searched, in the order they are loaded, to locate any additional dependencies. This process will continue until all the dependencies of <i>pathname</i> are loaded. This dependency tree is referred to as a <i>group</i>.</p> <p>If the value of <i>pathname</i> is 0, dlopen() provides a <i>handle</i> on a global symbol object. This object provides access to the symbols from an ordered set of objects consisting of the original program image file, together with any dependencies loaded at program startup, and any objects that were loaded using dlopen() together with the <code>RTLD_GLOBAL</code> flag. As the latter set of objects can change during process execution, the set identified by <i>handle</i> can also change dynamically.</p> |

dlopen() is identical to the **dlopen()** routine, except that an identifying link-map id (*lmid*) is passed into it. This link-map id informs the dynamic linking facilities upon which link-map list to load the object. See *Linker and Libraries Guide*

The *mode* parameter describes how **dlopen()** will operate upon *pathname* with respect to the processing of relocations and the scope of visibility of the symbols provided by *pathname* and its dependencies. When an object is brought into the address space of a process, it may contain references to symbols for which addresses are not known until the object is loaded. These references must be relocated before the symbols can be accessed. The *mode* parameter governs when these relocations take place and may have the following values:

RTLD_LAZY Only references to data symbols are relocated when the object is first loaded. References to functions are not relocated until a given function is invoked for the first time. This *mode* should improve performance, since a process may not reference all of the functions in any given object. This behavior mimics the normal loading of dependencies during process initialization.

RTLD_NOW All necessary relocations are performed when the object is first loaded. This may waste some processing, if relocations are performed for functions that are never referenced. This behavior may be useful for applications that need to know as soon as an object is loaded that all symbols referenced during execution will be available. This option mimics the loading of dependencies when the environment variable **LD_BIND_NOW** is in effect.

To determine the scope of visibility for symbols loaded with a **dlopen()** invocation, the *mode* parameter should be bitwise **OR**'ed with one of the following values:

RTLD_GLOBAL The object's global symbols are made available for the relocation processing of any other object. In addition, symbol lookup using **dlopen(0, mode)** and an associated **dlsym()**, allows objects loaded with **RTLD_GLOBAL** to be searched.

RTLD_LOCAL The object's global symbols are only available for the relocation processing of other objects that comprise the same group.

The program image file, and any objects loaded at program startup, have the mode **RTLD_GLOBAL**. The mode **RTLD_LOCAL** is the default mode for any objects acquired with **dlopen()**. A local object may be a dependency of more than one group. Any object of mode **RTLD_LOCAL** that is referenced as a dependency of an object of mode **RTLD_GLOBAL** will be promoted to **RTLD_GLOBAL**. In other words, the **RTLD_LOCAL** mode is ignored.

Any object loaded by **dlopen()** that requires relocations against global symbols can reference the symbols in any `RTLD_GLOBAL` object, which are at least the program image file and any objects loaded at program startup, from the object itself, and from any dependencies the object references. However, the *mode* parameter may also be bitwise or 'ed with the following values to affect the scope of symbol availability:

`RTLD_GROUP` Only symbols from the associated group are made available for relocation. A group is established from the defined object and all the dependencies of that object. A group must be completely self-contained. All dependency relationships between the members of the group must be sufficient to satisfy the relocation requirements of each object that comprises the group.

`RTLD_PARENT` The symbols of the object initiating the **dlopen()** call are made available to the objects obtained by **dlopen()** itself. This option is useful when hierarchical **dlopen()** families are created. Note that although the parent object can supply symbols for the relocation of this object, the parent object is not available to **dlsym()** through the returned *handle* .

`RTLD_WORLD` Only symbols from `RTLD_GLOBAL` objects are made available for relocation.

The default modes for **dlopen()** are both `RTLD_WORLD` and `RTLD_GROUP` . These modes are or 'ed together if an object is required by different dependencies specifying differing modes.

The following modes provide additional capabilities outside of relocation processing:

`RTLD_NODELETE` The specified object will not be deleted from the address space as part of a **dlclose()** .

`RTLD_NOLOAD` The specified object is not loaded as part of the **dlopen()** , but a valid *handle* is returned if the object already exists as part of the process address space. Additional modes can be specified and will be or 'ed with the present mode of the object and its dependencies. The `RTLD_NOLOAD` mode provides a means of querying the presence, or promoting the modes, of an existing dependency.

The *lmid* passed to **dlmopen()** identifies the link-map list where the object will be loaded. This can be any valid `Lmid_t` returned by **dlinfo()** or one of the following special values:

`LM_ID_BASE` Load the object on the applications link-map list.

LM_ID_LDSO Load the object on the dynamic linkers (`ld.so.1`) link-map list.

LM_ID_NEWLM Causes the object to create a new link-map list as part of loading. It is vital that any object opened on a new link-map list have all of its dependencies expressed because there will be no other objects on this link-map.

RETURN VALUES

If *pathname* cannot be found, cannot be opened for reading, is not a shared or relocatable object, or if an error occurs during the process of loading *pathname* or relocating its symbolic references, **dlopen()** will return NULL. More detailed diagnostic information will be available through **dlderror()** .

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

ld(1) , **ld.so.1(1)** , **dladdr(3X)** , **dlclose(3X)** , **dldump(3X)** , **dlderror(3X)** , **dlinfo(3X)** , **dlsym(3X)** , **attributes(5)**

Linker and Libraries Guide

NOTES

If other objects were link-edited with *pathname* when *pathname* was built, that is, the *pathname* has dependencies on other objects, those objects will automatically be loaded by **dlopen()** . The directory search path used to find both *pathname* and the other *needed* objects may be affected by setting the environment variable `LD_LIBRARY_PATH` , which is analyzed once at process startup, and from a `runpath` setting within the object from which the call to **dlopen()** originated. These search rules will only be applied to path names that do not contain an embedded `'/'` . Objects whose names resolve to the same absolute or relative path name may be opened any number of times using **dlopen()** ; however, the object referenced will only be loaded once into the address space of the current process.

When loading shared objects the application should open a specific version of the shared object, as opposed to relying on the version of the shared object pointed to by the symbolic link.

When building objects that are to be loaded on a new link-map list (see `LM_ID_NEWLM`), some precautions need to be taken. In general, all dependencies must be included when building an object. Also, include `/usr/lib/libmapmalloc.so.1` before `/usr/lib/libc.so.1` when building an object.

When an object is loaded into memory on a new link-map list, it is isolated from the main running program. There are certain global resources that are only usable from one link-map list. A few examples of these would be the **sbrk()** based **malloc()** , **libthread()** , and the signal vectors. Because of this, care must be taken not to use any of these resources on any but the primary link-map list. These issues are discussed in further detail in the *Linker and Libraries Guide*

Some symbols defined in dynamic executables or shared objects may not be available to the runtime linker. The symbol table created by `ld` for use by the runtime linker might contain only a subset of the symbols defined in the object.

| | |
|----------------------|--|
| NAME | dlsym – get the address of a symbol in a shared object |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -ldl [<i>library</i> ...] #include <dlfcn.h> void *dlsym(void *<i>handle</i>, const char *<i>name</i>);</pre> |
| DESCRIPTION | <p>dlsym() is one of a family of routines that give the user direct access to the dynamic linking facilities. (See <i>Linker and Libraries Guide</i>). These routines are made available via the library loaded when the option <code>-ldl</code> is passed to the link-editor.</p> <p>Note: <i>These routines are available to dynamically-linked processes ONLY.</i></p> <p>dlsym() allows a process to obtain the address of a symbol defined within a shared object. <i>handle</i> is either the value returned from a call to dlopen() or one of the special flags <code>RTLD_NEXT</code> or <code>RTLD_DEFAULT</code>. <i>name</i> is the symbol's name as a character string.</p> <p>In the case of a handle returned from dlopen() the corresponding shared object must not have been closed using dlclose(). dlsym() will search for the named symbol in all shared objects loaded automatically as a result of loading the object referenced by <i>handle</i>. See dlopen(3X).</p> <p>In the case of the special handle <code>RTLD_NEXT</code>, dlsym() will search for the named symbol in the objects that were loaded following the object from which the dlsym() call is being made.</p> <p>In the case of the special handle <code>RTLD_DEFAULT</code>, dlsym() will search for the named symbol, starting with the first object loaded and proceeding through the list of loaded objects until a match is found. This search follows the default model employed to relocate all objects within the process.</p> <p>In the case of both <code>RTLD_NEXT</code> and <code>RTLD_DEFAULT</code>, if the objects being searched have been loaded from dlopen() calls, dlsym() will search the object only if the caller is part of the same dlopen() dependency hierarchy, or if the object was given global search access. See dlopen(3X) for a discussion of the <code>RTLD_GLOBAL</code> mode.</p> |
| RETURN VALUES | If <i>handle</i> does not refer to a valid object opened by dlopen() , is not the special flag <code>RTLD_NEXT</code> , or if the named symbol cannot be found within any of the objects associated with <i>handle</i> , dlsym() will return <code>NULL</code> . More detailed diagnostic information is available through dlerror() . |
| EXAMPLES | <p>EXAMPLE 1 The use of dlopen() and dlsym().</p> <p>The following example shows how one can use dlopen() and dlsym() to access either function or data objects. For simplicity, error checking has been omitted.</p> |

```

void      *handle;
int       *iptr, (*fptr)(int);

/* open the needed object */
handle = dlopen("/usr/home/me/libfoo.so.1", RTLD_LAZY);

/* find the address of function and data objects */
fptr = (int (*)(int))dlsym(handle, "my_function");
iptr = (int *)dlsym(handle, "my_object");

/* invoke function, passing value of integer as a parameter */
(*fptr)(*iptr);

```

The following code fragment shows how **dlsym()** can be used to check to see that a particular function is defined and to call it only if it is.

```

int       (*fptr)();

if ((fptr = (int (*)())dlsym(RTLD_DEFAULT,
    "my_function")) != NULL) {
    (*fptr)();
}

```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

ld(1), **dladdr(3X)**, **dlclose(3X)**, **dldump(3X)**, **dlerror(3X)**, **dlopen(3X)**, **attributes(5)**

Linker and Libraries Guide

| | |
|--------------------|---|
| NAME | DmiAddComponent, DmiAddGroup, DmiAddLanguage, DmiDeleteComponent, DmiDeleteGroup, DmiDeleteLanguage – Management Interface database administration functions |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -ldmimi -ldmi -lnsl -lrwtool [<i>library</i> ...] #include <server.h> #include <miapi.h> bool_t DmiAddComponent(DmiAddComponentIN <i>argin</i>, DmiAddComponentOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiAddGroup(DmiAddGroupIN <i>argin</i>, DmiAddGroupOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiAddLanguage(DmiAddLanguageIN <i>argin</i>, DmiAddLanguageOUT* <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiDeleteComponent(DmiDeleteComponentIN <i>argin</i>, DmiDeleteComponentOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiDeleteGroup(DmiDeleteGroupIN <i>argin</i>, DmiDeleteGroupOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiDeleteLanguage(DmiDeleteLanguageIN <i>argin</i>, DmiDeleteLanguageOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); </pre> |
| DESCRIPTION | <p>The database administration functions add a new component to the database or add a new language mapping for an existing component. You may also remove an existing component, remove a specific language mapping, or remove a group from a component.</p> <p>The DmiAddComponent() function adds a new component to the DMI database. It takes the name of a file, or the address of memory block containing</p> |

MIF data, checks the data for adherence to the DMI MIF grammar, and installs the MIF in the database. The procedure returns a unique component ID for the newly installed component. The *argIn* parameter is an instance of a `DmiAddComponentIN` structure containing the following members:

```
DmiHandle_t      handle;          /* an open session handle */
DmiFileDataList_t *fileData;     /* MIF data for component */
```

The *result* parameter is a pointer to a `DmiAddComponentOUT` structure containing the following members:

```
DmiErrorStatus_t error_status;
DmiId_t          compId;         /* SP-allocated component ID */
DmiStringList_t *errors;        /* installation error messages */
```

The **DmiAddLanguage()** function adds a new language mapping for an existing component in the database. It takes the name of a file, or the address of memory block containing translated MIF data, checks the data for adherence to the DMI MIF grammar, and installs the language MIF in the database. The *argIn* parameter is an instance of a `DmiAddLanguageIN` structure containing the following members:

```
DmiHandle_t      handle;          /* an open session handle */
DmiFileDataList_t *fileData;     /* language mapping file */
DmiId_t          compId;         /* component to access */
```

The *result* parameter is a pointer to a `DmiAddLanguageOUT` structure containing the following members:

```
DmiErrorStatus_t error_status;
DmiStringList_t *errors;         /* installation error messages */
```

The **DmiAddGroup()** function adds a new group to an existing component in the database. It takes the name of a file, or the address of memory block containing the group's MIF data, checks the data for adherence to the DMI MIF grammar, and installs the group MIF in the database. The *argIn* parameter is an instance of a `DmiAddGroupIN` structure containing the following members:

```
DmiHandle_t      handle;          /* an open session handle */
DmiFileDataList_t *fileData;     /* MIF file data for group */
```

```
DmiId_t          compId;          /* component to access */
```

The *result* parameter is a pointer to a `DmiAddGroupOUT` structure containing the following members:

```
DmiErrorStatus_t  error_status;
DmiId_t           groupId;       /* SP-allocated group ID */
DmiStringList_t  *errors;       /* installation error messages */
```

The **DmiDeleteComponent()** function removes an existing component from the database. The *argIn* parameter is an instance of a `DmiDeleteComponentIN` structure containing the following members:

```
DmiHandle_t       handle;        /* an open session handle */
DmiId_t           compId;        /* component to delete */
```

The *result* parameter is a pointer to a `DmiDeleteComponentOUT` structure containing the following members:

```
DmiErrorStatus_t  error_status;
```

The **DmiDeleteLanguage()** function removes a specific language mapping for a component. You specify the language string and component ID. The *argIn* parameter is an instance of a `DmiDeleteLanguageIN` structure containing the following members:

```
DmiHandle_t       handle;        /* an open session handle */
DmiString_t       *language;     /* language to delete */
DmiId_t           compId;        /* component to access */
```

The *result* parameter is a pointer to a `DmiDeleteLanguageOUT` structure containing the following members:

```
DmiErrorStatus_t  error_status;
```

The **DmiDeleteGroup()** function removes a group from a component. The caller specifies the component and group IDs. The *argIn* parameter is an instance of a `DmiDeleteGroupIN` structure containing the following members:

```

DmiHandle_t      handle;      /* an open session handle */
DmiId_t          compId;      /* component containing group */
DmiId_t          groupId;     /* group to delete */

```

The *result* parameter is a pointer to a `DmiDeleteGroupOUT` structure containing the following members:

```

DmiErrorStatus_t  error_status;

```

RETURN VALUES

The **DmiAddComponent()** function returns the following possible values:

```

DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_FILE_ERROR
DMIERR_BAD_SCHEMA_DESCRIPTION_FILE

```

The **DmiAddGroup()** function returns the following possible values:

```

DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE

DMIERR_INSUFFICIENT_PRIVILEGES
DMIERR_COMPONENT_NOT_FOUND
DMIERR_FILE_ERROR
DMIERR_BAD_SCHEMA_DESCRIPTION_FILE

```

The **DmiAddLanguage()** function returns the following possible values:

```

DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_COMPONENT_NOT_FOUND
DMIERR_FILE_ERROR
DMIERR_BAD_SCHEMA_DESCRIPTION_FILE

```

The **DmiDeleteComponent()** function returns the following possible values:

```

DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_INSUFFICIENT_PRIVILEGES
DMIERR_COMPONENT_NOT_FOUND
DMIERR_FILE_ERROR

```

The **DmiDeleteGroup()** function returns the following possible values:

```

DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_INSUFFICIENT_PRIVILEGES
DMIERR_COMPONENT_NOT_FOUND
DMIERR_FILE_ERROR

```

The **DmiDeleteLanguage()** function returns the following possible values:

```

DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_COMPONENT_NOT_FOUND
DMIERR_FILE_ERROR

```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-level | Unsafe |

SEE ALSO

attributes(5)

| | |
|--------------------|--|
| NAME | DmiAddRow, DmiDeleteRow, DmiGetAttribute, DmiGetMultiple, DmiSetAttribute, DmiSetMultiple – Management Interface operation functions |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -ldmimi -ldmi -lnsl -lrwtool [<i>library</i> ...] #include <server.h> #include <miapi.h> bool_t DmiAddRow(DmiAddRowIN <i>argin</i>, DmiAddRowOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiDeleteRow(DmiDeleteRowIN <i>argin</i>, DmiDeleteRowOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiGetAttribute(DmiGetAttributeIN <i>argin</i>, DmiGetAttributeOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiGetMultiple(DmiGetMultipleIN <i>argin</i>, DmiGetMultipleOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiSetAttribute(DmiSetAttributeIN <i>argin</i>, DmiSetAttributeOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiSetMultiple(DmiSetMultipleIN <i>argin</i>, DmiSetMultipleOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); </pre> |
| DESCRIPTION | <p>The operation functions provide a method for retrieving a single value from the Service Provider and for setting a single attribute value. In addition, you may also retrieve attribute values from the Service Provider. You may perform a set operation on an attribute or a list of attributes and add or delete a row from an existing table.</p> <p>The DmiAddRow() function adds a row to an existing table. The <code>rowData</code> parameter contains the full data, including key attribute values, for a row. It is</p> |

an error for the key list to specify an existing table row. The *argIn* parameter is an instance of a DmiAddRowIN structure containing the following members:

```
DmiHandle_t      handle;      /* An open session handle */
DmiRowData_t    *rowData;    /* Attribute values to set */
```

The *result* parameter is a pointer to a DmiAddRowOUT structure containing the following members:

```
DmiErrorStatus_t  error_status;
```

DmiDeleteRow() function removes a row from an existing table. The key list must specify valid keys for a table row. The *argIn* parameter is an instance of a DmiDeleteRowIN structure containing the following members:

```
\011
DmiHandle_t      handle;      /* An open session handle */
DmiRowData_t    *rowData;    /* Row to delete */
```

The *result* parameter is a pointer to a DmiDeleteRowOUT structure containing the following members:

```
DmiErrorStatus_t  error_status;
```

The **DmiGetAttribute()** function provides a simple method for retrieving a single attribute value from the Service Provider. The *compId*, *groupId*, *attribId*, and *keyList* identify the desired attribute. The resulting attribute value is returned in a newly allocated DmiDataUnion structure. The address of this structure is returned through the *value* parameter. The *argIn* parameter is an instance of a DmiListComponentsIN structure containing the following members:

```
DmiHandle_t      handle;      /* an open session handle */
DmiId_t          compId;      /* Component to access */
DmiId_t          groupId;    /* Group within component */
DmiId_t          attribId;   /* Attribute within a group */
DmiAttributeValues_t
*keyList;        /* Keylist to specify a table row */
```

The *result* parameter is a pointer to a DmiGetAttributeOUT structure containing the following members:

```
DmiErrorStatus_t    error_status;
DmiDataUnion_t     *value;      /* Attribute value returned */
```

The **DmiGetMultiple()** function retrieves attribute values from the Service Provider. This procedure may get the value for an individual attribute, or for multiple attributes across groups, components, or rows of a table.

The **DmiSetAttribute()** function provides a simple method for setting a single attribute value. The `compId`, `groupId`, `attribId`, and `keyList` identify the desired attribute. The `setMode` parameter defines the procedure call as a Set, Reserve, or Release operation. The new attribute value is contained in the `DmiDataUnion` structure whose address is passed in the `value` parameter. The *argIn* parameter is an instance of a `DmiSetAttributeIN` structure containing the following members:

```
DmiHandle_t        handle;
DmiId_t            compId;
DmiId_t            groupId;
DmiId_t            attribId;
DmiAttributeValues_t *keyList;
DmiSetMode_t       setMode;
DmiDataUnion_t     *value;
```

The *result* parameter is a pointer to a `DmiSetAttributeOUT` structure containing the following members:

```
DmiErrorStatus_t    error_status;
```

The **DmiSetMultiple()** function performs a set operation on an attribute or list of attributes. Set operations include actually setting the value, testing and reserving the attribute for future setting, or releasing the set reserve. These variations on the set operation are specified by the parameter `setMode`. The *argIn* parameter is an instance of a `DmiSetMultipleIN` structure containing the following members:

```
DmiHandle_t        handle;      /* An open session handle */
DmiSetMode_t       setMode;     /* set, reserve, or release */
DmiMultiRowData_t *rowData;     /* Attribute values to set */
```

The *result* parameter is a pointer to a `DmiSetMultipleOUT` structure containing the following members:

```
DmiErrorStatus_t    error_status;
```

The `rowData` array describes the attributes to set, and contains the new attribute values. Each element of `rowData` specifies a component, group, key list (for table accesses), and attribute list to set. No data is returned from this function.

RETURN VALUES

The **DmiAddRow()** function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_VALUE_UNKNOWN
DMIERR_COMPONENT_NOT_FOUND
DMIERR_GROUP_NOT_FOUND
DMIERR_ILLEGAL_KEYS
DMIERR_DIRECT_INTERFACE_NOT_REGISTERED
DMIERR_UNKNOWN_CI_REGISTRY
DMIERR_VALUE_UNKNOWN
DMIERR_UNABLE_TO_ADD_ROW
```

The **DmiDeleteRow()** function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_ATTRIBUTE_NOT_FOUND
DMIERR_COMPONENT_NOT_FOUND
DMIERR_GROUP_NOT_FOUND
DMIERR_ILLEGAL_KEYS
DMIERR_ILLEGAL_TO_GET
DMIERR_DIRECT_INTERFACE_NOT_REGISTERED
DMIERR_ROW_NOT_FOUND
DMIERR_UNKNOWN_CI_REGISTRY
DMIERR_VALUE_UNKNOWN
DMIERR_UNABLE_TO_DELETE_ROW
```

The **DmiGetAttribute()** function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_ATTRIBUTE_NOT_FOUND
```

```

DMIERR_COMPONENT_NOT_FOUND
DMIERR_GROUP_NOT_FOUND
DMIERR_ILLEGAL_KEYS
DMIERR_ILLEGAL_TO_GET
DMIERR_DIRECT_INTERFACE_NOT_REGISTERED
DMIERR_ROW_NOT_FOUND
DMIERR_UNKNOWN_CI_REGISTRY
DMIERR_FILE_ERROR
DMIERR_VALUE_UNKNOWN

```

The **DmiGetMultiple()** function returns the following possible values:

```

DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_RPC_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_ATTRIBUTE_NOT_FOUND
DMIERR_COMPONENT_NOT_FOUND
DMIERR_GROUP_NOT_FOUND
DMIERR_ILLEGAL_KEYS
DMIERR_ILLEGAL_TO_GET
DMIERR_DIRECT_INTERFACE_NOT_REGISTERED
DMIERR_ROW_NOT_FOUND
DMIERR_UNKNOWN_CI_REGISTRY
DMIERR_FILE_ERROR
DMIERR_VALUE_UNKNOWN

```

The **DmiSetAttribute()** function returns the following possible values:

```

DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_ATTRIBUTE_NOT_FOUND
DMIERR_COMPONENT_NOT_FOUND
DMIERR_GROUP_NOT_FOUND
DMIERR_ILLEGAL_KEYS
DMIERR_ILLEGAL_TO_GET
DMIERR_DIRECT_INTERFACE_NOT_REGISTERED
DMIERR_ROW_NOT_FOUND
DMIERR_UNKNOWN_CI_REGISTRY
DMIERR_FILE_ERROR
DMIERR_VALUE_UNKNOWN

```

The **DmiSetMultiple()** function returns the following possible values:

```

DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY

```

```

DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_ATTRIBUTE_NOT_FOUND
DMIERR_COMPONENT_NOT_FOUND
DMIERR_GROUP_NOT_FOUND
DMIERR_ILLEGAL_KEYS
DMIERR_ILLEGAL_TO_SET
DMIERR_DIRECT_INTERFACE_NOT_REGISTERED
DMIERR_ROW_NOT_FOUND
DMIERR_UNKNOWN_CI_REGISTRY
DMIERR_FILE_ERROR
DMIERR_VALUE_UNKNOWN

```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-level | Unsafe |

SEE ALSO

attributes(5)

| | |
|--------------------|--|
| NAME | DmiGetConfig, DmiGetVersion, DmiRegister, DmiSetConfig, DmiUnregister – Management Interface initialization functions |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -ldmimi -ldmi -lnsl -lrwtool [<i>library</i> ...] #include <server.h> #include <miapi.h> bool_t DmiGetConfig(DmiGetConfigIN <i>argin</i>, DmiGetConfigOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiGetVersion(DmiGetVersionIN <i>argin</i>, DmiGetVersionOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiRegister(DmiRegisterIN <i>argin</i>, DmiRegisterOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiSetConfig(DmiSetConfigIN <i>argin</i>, DmiSetConfigOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiUnregister(DmiUnregisterIN <i>argin</i>, DmiUnregisterOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); </pre> |
| DESCRIPTION | <p>The Management Interface initialization functions enable you to register management applications to the Service Provider. You may also retrieve information about the Service Provider, get and set session configuration information for your session.</p> <p>The DmiGetConfig() function retrieves the per-session configuration information. The configuration information consists of a string describing the current language being used for the session. The <i>argin</i> parameter is an instance of a DmiGetConfigIN structure containing the following member:</p> <pre> DmiHandle_t handle; /* an open session handle */ </pre> |

The *result* parameter is a pointer to a `DmiGetConfigOUT` structure containing the following members:

```
DmiErrorStatus_t  error_status;
DmiString_t       *language;      /* current session language */
```

The **DmiGetVersion()** function retrieves information about the Service Provider. The management application uses the **DmiGetVersion()** procedure to determine the DMI specification level supported by the Service Provider. This procedure also returns the service provided description string, and may contain version information about the Service Provider implementation. The *argIn* parameter is an instance of a `DmiGetVersionIN` structure containing the following member:

```
DmiHandle_t       handle;         /* an open session handle */
```

The *result* parameter is a pointer to a `DmiGetVersionOUT` structure containing the following members:

```
DmiErrorStatus_t  error_status;
DmiString_t       *dmiSpecLevel; /* DMI specification version */
DmiString_t       *description;   /* OS specific DMI SP version */
DmiFileTypeList_t *fileTypes;     /* file types for MIF installation */
```

The **DmiRegister()** function provides the management application with a unique per-session handle. The Service Provider uses this procedure to initialize to an internal state for subsequent procedure calls made by the application. This procedure must be the first command executed by the management application. *argIn* is an instance of a `DmiRegisterIN` structure containing the following member:

```
DmiHandle_t       handle;         /* an open session handle */
```

The *result* parameter is a pointer to a `DmiRegisterOUT` structure containing the following members:

```
DmiErrorStatus_t  error_status;
DmiHandle_t       *handle;        /* an open session handle */
```

The **DmiSetConfig()** function sets the per-session configuration information. The configuration information consists of a string describing the language required by the management application. The *argIn* parameter is an instance of a `DmiSetConfigIN` structure containing the following member:

```
DmiHandle_t      handle;          /* an open session handle */
DmiString_t     *language;      /* current language required */
```

The *result* parameter is a pointer to a `DmiSetConfigOUT` structure containing the following member:

```
DmiErrorStatus_t error_status;
```

The **DmiUnregister()** function is used by the Service Provider to perform end-of-session cleanup actions. On return from this function, the session handle is no longer valid. This function must be the last DMI command executed by the management application. The *argIn* parameter is an instance of a `DmiUnregisterIN` structure containing the following member:

```
DmiHandle_t      handle;          /* an open session handle */
```

The *result* parameter is a pointer to a `DmiUnregisterOUT` structure containing the following members:

```
DmiErrorStatus_t error_status;
```

RETURN VALUES

The **DmiGetConfig()** function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
```

The **DmiGetVersion()** function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
```

DMIERR_OUT_OF_MEMORY
 DMIERR_SP_INACTIVE

The **DmiRegister()** function returns the following possible values:

DMIERR_NO_ERROR
 DMIERR_ILLEGAL_RPC_HANDLE
 DMIERR_OUT_OF_MEMORY
 DMIERR_SP_INACTIVE

The **DmiSetConfig()** function returns the following possible values:

DMIERR_NO_ERROR
 DMIERR_ILLEGAL_RPC_HANDLE
 DMIERR_OUT_OF_MEMORY
 DMIERR_ILLEGAL_PARAMETER
 DMIERR_SP_INACTIVE
 DMIERR_ILLEGAL_TO_SET

The **DmiUnRegister()** function returns the following possible values:

DMIERR_NO_ERROR
 DMIERR_ILLEGAL_RPC_HANDLE
 DMIERR_OUT_OF_MEMORY
 DMIERR_ILLEGAL_PARAMETER
 DMIERR_SP_INACTIVE

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-level | Unsafe |

SEE ALSO

attributes(5)

| | |
|--------------------|---|
| NAME | DmiListAttributes, DmiListClassNames, DmiListComponents, DmiListComponentsByClass, DmiListGroups, DmiListLanguages – Management Interface listing functions |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -ldmimi -ldmi -lnsl -lrwtool [<i>library</i> ...] #include <server.h> #include <miapi.h> bool_t DmiListAttributes(DmiListAttributesIN <i>argin</i>, DmiListAttributesOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiListClassNames(DmiListClassNamesIN <i>argin</i>, DmiListClassNamesOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiListComponents(DmiListComponentsIN <i>argin</i>, DmiListComponentsOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiListComponentsByClass(DmiListComponentsByClassIN <i>argin</i>, DmiListComponentsByClassOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiListGroups(DmiListGroupsIN <i>argin</i>, DmiListGroupsOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiListLanguages(DmiListLanguagesIN <i>argin</i>, DmiListLanguagesOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); </pre> |
| DESCRIPTION | <p>The listing functions enables you to retrieve the names and the description of components in a system. You may also list components by class that match a specified criteria. The listing functions retrieve the set of language mappings installed for a specified component, retrieve class name strings for all groups in a component, retrieve a list of groups within a component, and retrieve the properties for one or more attributes in a group.</p> |

The **DmiListComponents()** function retrieves the name and (optionally) the description of components in a system. Use this to interrogate a system to determine what components are installed. The *argin* parameter is an instance of a `DmiListComponentsIN` structure containing the following members:

```
DmiHandle_t      handle;          /* an open session handle */
DmiRequestMode_t requestMode;    /* Unique, first, or next */
DmiUnsigned_t   maxCount;        /* maximum number to return,
                                0 for all */
DmiBoolean_t    getPragma;       /* get optional pragma string */
DmiBoolean_t    getDescription; /* get optional component
                                description */
DmiId_t         compId;          /* component ID to start with */
```

The *result* parameter is a pointer to a `DmiListComponentsOUT` structure containing the following members:

```
DmiErrorStatus_t error_status;
DmiComponentList_t *reply;      /* list of components */
```

An enumeration accesses a specific component or may be used to sequentially access all components in a system. The caller may choose not to retrieve the component description by setting the value `getDescription` to false. The caller may choose not to retrieve the pragma string by setting the value of `getta-percha` to false. The `maxCount`, `requestMode`, and `compId` parameters allow the caller to control the information returned by the Service Provider. When the `requestMode` is `DMI_UNIQUE`, `compId` specifies the first component requested (or only component if `maxCount` is one). When the `requestMode` is `DMI_NEXT`, `compId` specifies the component just before the one requested. When `requestMode` is `DMI_FIRST`, `compId` is unused.

To control the amount of information returned, the caller sets `maxCount` to something other than zero. The service provider must honor this limit on the amount of information returned. When `maxCount` is 0 the service provider returns information for all components, subject to the constraints imposed by `requestMode` and `compId`.

The **DmiListComponentsByClass()** function lists components that match specified criteria. Use this function to determine if a component contains a certain group or a certain row in a table. A filter condition may be that a component contains a specified group class name or that it contains a specific row in a specific group. As with **DmiListComponents()**, the description and pragma strings are optional return values. *argin* is an instance of a `DmiListComponentsByClassIN` structure containing the following members:

```

DmiHandle_t          handle;          /* an open session handle */
DmiRequestMode_t    requestMode;     /* Unique, first or next */
DmiUnsigned_t       maxCount;        /* maximum number to return,
                                     or 0 for all */
DmiBoolean_t        getPragma;       /* get the optional pragma
                                     string */
DmiBoolean_t        getDescription;  /* get optional component
                                     description */
DmiId_t             compId;          /* component ID to start with */
DmiString_t         *className;      /* group class name string
                                     to match*/
DmiAttributeValues_t *keyList;       /* group row keys to match */

```

The *result* parameter is a pointer to a `DmiListComponentsbyClassOUT` structure containing the following members:

```

DmiErrorStatus_t    error_status;
DmiComponentList_t *reply;          /* list of components */

```

The `DmiListLanguages()` function retrieves the set of language mappings installed for the specified component. The *argIn* parameter is an instance of a `DmiListLanguagesIN` structure containing the following members:

```

DmiHandle_t          handle;          /* An open session handle */
DmiUnsigned_t       maxCount;        /* maximum number to return,
                                     or 0 for all */
DmiId_t             compId;          /* Component to access */

```

The *result* parameter is a pointer to a `DmiListLanguagesOUT` structure containing the following members:

```

DmiErrorStatus_t    error_status;
DmiStringList_t     *reply;          /* List of language strings */

```

The `DmiListClassNames()` function retrieves the class name strings for all groups in a component. This enables the management application to easily determine if a component contains a specific group, or groups. The *argIn* parameter is an instance of a `DmiListClassNamesIN` structure containing the following members:

```

DmiHandle_t          handle;          /* An open session handle */
DmiUnsigned_t       maxCount;        /* maximum number to return,

```

```
DmiId_t          compId;          /* Component to access */
                                     or 0 for all */
```

The *result* parameter is a pointer to a `DmiListClassNamesOUT` structure containing the following members:

```
DmiErrorStatus_t  error_status;
DmiClassNameList_t *reply;        /* List of class names and
group IDs */
```

The `DmiListGroups()` function retrieves a list of groups within a component. With this function you can access a specific group or sequentially access all groups in a component. All enumerations of groups occur within the specified component and do not span components. The *argIn* parameter is an instance of a `DmiListGroupsIN` structure containing the following members:

```
DmiHandle_t       handle;         /* An open session handle */
DmiRequestMode_t  requestMode;    /* Unique, first or next group */
DmiUnsigned_t     maxCount;       /* Maximum number to return,
                                     or 0 for all */
DmiBoolean_t      getPragma;      /* Get the optional pragma string */
DmiBoolean_t      getDescription; /* Get optional group description */
DmiId_t           compId;         /* Component to access */
DmiId_t           groupId;        /* Group to start with, refer to
                                     requestMode */
```

The *result* parameter is a pointer to a `DmiListGroupsOUT` structure containing the following members:

```
DmiErrorStatus_t  error_status;
DmiGroupList_t    *reply;\011
```

The caller may choose not to retrieve the group description by setting the value `getDescription` to false. The caller may choose not to retrieve the pragma string by setting the value of `getPragma` to false. The `maxCount`, `requestMode`, and `groupId` parameters allow the caller to control the information returned by the Service Provider. When the `requestMode` is `DMI_UNIQUE`, `groupId` specifies the first group requested (or only group if `maxCount` is one). When the `requestMode` is `DMI_NEXT`, `groupId` specifies the group just before the one requested. When `requestMode` is `DMI_FIRST`, `groupId` is unused. To control the amount of information returned, the caller

sets `maxCount` to something other than zero. The service provider must honor this limit on the amount of information returned. When `maxCount` is zero the service provider returns information for all groups, subject to the constraints imposed by `requestMode` and `groupId`.

The **DmiListAttributes()** function retrieves the properties for one or more attributes in a group. All enumerations of attributes occur within the specified group, and do not span groups. The *argin* parameter is an instance of a `DmiListAttributesIN` structure containing the following members:

```
DmiHandle_t      handle;          /* An open session handle */
DmiRequestMode_t requestMode;    /* Unique, first or next group */
DmiUnsigned_t   maxCount;       /* Maximum number to return,
                                or 0 for all */
DmiBoolean_t    getPragma;      /* Get the optional pragma string */
DmiBoolean_t    getDescription; /* Get optional group description */
DmiId_t         compId;         /* Component to access */
DmiId_t         groupId;        /* Group to access */
DmiId_t         attribId;       /* Attribute to start with, refer
                                to requestMode */
```

The *result* parameter is a pointer to a `DmiListAttributesOUT` structure containing the following members:

```
DmiErrorStatus_t error_status;
DmiAttributeList_t *reply;      /* List of attributes */
```

You may choose not to retrieve the description string by setting the value of `getDescription` to false. Likewise, you may choose not to retrieve the pragma string by setting the value of `getPragma` to false. The `maxCount`, `requestMode`, and `attribId` parameters allow you to control the information returned by the Service Provider. When the `requestMode` is `DMI_UNIQUE`, `attribId` specifies the first attribute requested (or only attribute if `maxCount` is one). When the `requestMode` is `DMI_NEXT`, `attribId` specifies the attribute just before the one requested. When `requestMode` is `DMI_FIRST`, `attribId` is unused. To control the amount of information returned, the caller sets `maxCount` to something other than zero. The Service Provider must honor this limit on the amount of information returned. When `maxCount` is zero the service provider returns information for all attributes, subject to the constraints imposed by `requestMode` and `attribId`.

RETURN VALUES

The **DmiListAttributes()** function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
```

```
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_ATTRIBUTE_NOT_FOUND
DMIERR_COMPONENT_NOT_FOUND
DMIERR_GROUP_NOT_FOUND
DMIERR_FILE_ERROR
```

The **DmiListClassNames()** function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_COMPONENT_NOT_FOUND
DMIERR_FILE_ERROR
```

The **DmiListComponents()** function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_COMPONENT_NOT_FOUND
DMIERR_FILE_ERROR
```

The **DmiListComponentsByClass()** function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_COMPONENT_NOT_FOUND
DMIERR_FILE_ERROR
```

The **DmiListGroups()** function returns the following possible values:

```
DMIERR_NO_ERROR
DMIERR_ILLEGAL_RPC_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_PARAMETER
DMIERR_SP_INACTIVE
DMIERR_COMPONENT_NOT_FOUND
DMIERR_GROUP_NOT_FOUND
DMIERR_FILE_ERROR
```

The **DmiListLanguages()** function returns the following possible values:

```
DMIERR_NO_ERROR  
DMIERR_ILLEGAL_RPC_HANDLE  
DMIERR_OUT_OF_MEMORY  
DMIERR_ILLEGAL_PARAMETER  
DMIERR_SP_INACTIVE  
DMIERR_COMPONENT_NOT_FOUND  
DMIERR_FILE_ERROR
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-level | Unsafe |

SEE ALSO

attributes(5)

| | |
|--------------------|--|
| NAME | DmiRegisterCi, DmiUnRegisterCi, DmiOriginateEvent – Service Provider functions for components |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lci -ldmi -lnsl -lrwtool [<i>library</i> ...] #include <server.h> #include <ciapi.h> extern bool_t DmiRegisterCi(DmiRegisterCiIN <i>argin</i>, DmiRegisterCiOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiUnregisterCi(DmiUnregisterCiIN <i>argin</i>, DmiUnregisterCiOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>); bool_t DmiOriginateEvent(DmiOriginateEventIN <i>argin</i>, DmiOriginateEventOUT * <i>result</i>, DmiRpcHandle * <i>dmi_rpc_handle</i>);</pre> |
| DESCRIPTION | <p>These three functions provide component communication with the DMI through the Component Interface (CI).</p> <p>Component instrumentation code may register with the Service Provider to override its current mechanism for the registered attributes. Instead of manipulating the data in the MIF database or invoking programs, the Service Provider calls the entry points provided in the registration call. Once the component unregisters, the Service Provider returns to a normal method of processing requests for the data as defined in the MIF. Component instrumentation can temporarily interrupt normal processing to perform special functions.</p> <p>Registering attributes through the direct interface overrides attributes that are already being served through the direct interface. RPC is used for communication from the Service Provider to the component instrumentation.</p> |

For all three functions, *argin* is the parameter passed to initiate an RPC call, *result* is the result of the RPC call, and *dmi_rpc_handle* is an open session RPC handle.

The **DmiRegisterCi()** function registers a callable interface for components that have resident instrumentation code and/or to get the version of the Service Provider.

The **DmiUnRegisterCi()** function communicates to the Service Provider to remove a direct component instrumentation interface from the Service Provider table of registered interfaces.

The **DmiOriginateEvent()** function originates an event for filtering and delivery. Any necessary indication filtering is performed by this function (or by subsequent processing) before the event is forwarded to the management applications.

A component ID value of zero (0) specifies the event was generated by something that has not been installed as a component, and has no component ID.

RETURN VALUES

The **DmiRegisterCi()** function returns the following possible values:

```

DMIERR_NO_ERROR
DMIERR_ILLEGAL_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_INSUFFICIENT_PRIVILEGES
DMIERR_SP_INACTIVE
DMIERR_ATTRIBUTE_NOT_FOUND
DMIERR_COMPONENT_NOT_FOUND
DMIERR_GROUP_NOT_FOUND
DMIERR_DATABASE_CORRUPT
DMIERR_OUT_OF_MEMORY
DMIERR_ILLEGAL_DMI_LEVEL

```

The **DmiUnRegisterCi()** function returns the following possible values:

```

DMIERR_NO_ERROR
DMIERR_ILLEGAL_HANDLE
DMIERR_OUT_OF_MEMORY
DMIERR_INSUFFICIENT_PRIVILEGES
DMIERR_SP_INACTIVE
DMIERR_UNKNOWN_CI_REGISTRY

```

The **DmiOriginateEvent()** function returns the following possible values:

```

DMIERR_NO_ERROR
DMIERR_ILLEGAL_HANDLE
DMIERR_OUT_OF_MEMORY

```

DMIERR_INSUFFICIENT_PRIVILEGES
DMIERR_SP_INACTIVE
DMIERR_UNKNOWN_CI_REGISTRY

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-level | Unsafe |

SEE ALSO

attributes(5)

| | |
|--------------------|---|
| NAME | doconfig – execute a configuration script |
| SYNOPSIS | <pre>cc [flag ...] file ... -lnsl [library ...] # include <sac.h></pre> <pre>int doconfig(int <i>filde</i>, char *<i>script</i>, long <i>rflag</i>);</pre> |
| DESCRIPTION | <p>doconfig() is a Service Access Facility library function that interprets the configuration scripts contained in the files <code></etc/saf/<i>pmtag</i>/_config></code>, <code></etc/saf/_sysconfig></code>, and <code></etc/saf/<i>pmtag</i>/<i>svctag</i>></code>, where <i>pmtag</i> specifies the tag associated with the port monitor, and <i>svctag</i> specifies the service tag associated with a given service. See <code>pmadm(1M)</code> and <code>sacadm(1M)</code>.</p> <p><i>script</i> is the name of the configuration script; <i>filde</i> is a file descriptor that designates the stream to which stream manipulation operations are to be applied; <i>rflag</i> is a bitmask that indicates the mode in which <i>script</i> is to be interpreted. If <i>rflag</i> is zero, all commands in the configuration script are eligible to be interpreted. If <i>rflag</i> has the NOASSIGN bit set, the <code>assign</code> command is considered illegal and will generate an error return. If <i>rflag</i> has the NORUN bit set, the <code>run</code> and <code>runwait</code> commands are considered illegal and will generate error returns.</p> <p>The configuration language in which <i>script</i> is written consists of a sequence of commands, each of which is interpreted separately. The following reserved keywords are defined: <code>assign</code>, <code>push</code>, <code>pop</code>, <code>runwait</code>, and <code>run</code>. The comment character is <code>#</code>; when a <code>#</code> occurs on a line, everything from that point to the end of the line is ignored. Blank lines are not significant. No line in a command script may exceed 1024 characters.</p> <p><code>assign <i>variable</i>=<i>value</i></code></p> <p>Used to define environment variables. <i>variable</i> is the name of the environment variable and <i>value</i> is the value to be assigned to it. The value assigned must be a string constant; no form of parameter substitution is available. <i>value</i> may be quoted. The quoting rules are those used by the shell for defining environment variables. <code>assign</code> will fail if space cannot be allocated for the new variable or if any part of the specification is invalid.</p> <p><code>push <i>module1</i>[, <i>module2</i>, <i>module3</i>, . . .]</code></p> <p>Used to push STREAMS modules onto the stream designated by <i>filde</i>. <i>module1</i> is the name of the first module to be pushed, <i>module2</i> is the name of the second module to be pushed, etc. The command will fail if any of the named modules cannot be pushed. If a module cannot be pushed, the subsequent modules on the same command line will be ignored and modules that have already been pushed will be popped.</p> |

pop [*module*]

Used to pop STREAMS modules off the designated stream. If `pop` is invoked with no arguments, the top module on the stream is popped. If an argument is given, modules will be popped one at a time until the named module is at the top of the stream. If the named module is not on the designated stream, the stream is left as it was and the command fails. If *module* is the special keyword ALL, then all modules on the stream will be popped. Note that only modules above the topmost driver are affected.

runwait command

The `runwait` command runs a command and waits for it to complete. `command` is the pathname of the command to be run. The command is run with `/usr/bin/sh -c` prepended to it; shell scripts may thus be executed from configuration scripts. The `runwait` command will fail if `command` cannot be found or cannot be executed, or if `command` exits with a non-zero status.

run command

The `run` command is identical to `runwait` except that it does not wait for `command` to complete. `command` is the pathname of the command to be run. `run` will not fail unless it is unable to create a child process to execute the command.

Although they are syntactically indistinguishable, some of the commands available to `run` and `runwait` are interpreter built-in commands. Interpreter built-ins are used when it is necessary to alter the state of a process within the context of that process. The **doconfig()** interpreter built-in commands are similar to the shell special commands and, like these, they do not spawn another process for execution. See `sh(1)`. The built-in commands are:

```
cd
ulimit
umask
```

RETURN VALUES

doconfig() returns 0 if the script was interpreted successfully. If a command in the script fails, the interpretation of the script ceases at that point and a positive number is returned; this number indicates which line in the script failed. If a system error occurs, a value of -1 is returned. When a script fails, the process whose environment was being established should *not* be started.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `sh(1)`, `pmadm(1M)`, `sacadm(1M)`, `attributes(5)`

NOTES This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

| | |
|--------------------|---|
| NAME | door_bind, door_unbind – bind or unbind the current thread with the door server pool |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -ldoor -lthread [<i>library</i> ...] #include <door.h> int door_bind(int <i>did</i>); int door_unbind();</pre> |
| DESCRIPTION | <p>door_bind() associates the current thread with a door server pool. A door server pool is a private pool of server threads that is available to serve door invocations associated with the door <i>did</i> .</p> <p>door_unbind() breaks the association of door_bind() by removing any private door pool binding that is associated with the current thread.</p> <p>Normally, door server threads are placed in a global pool of available threads that invocations on any door can use to dispatch a door invocation. A door that has been created with <code>DOOR_PRIVATE</code> only uses server threads that have been associated with the door by door_bind() . Therefore, it is necessary to bind at least one server thread to doors created with <code>DOOR_PRIVATE</code> .</p> <p>The server thread create routine, door_server_create() , is initially called by the system during a door_create() operation. See door_server_create(3X) and door_create(3X) .</p> <p>The current thread is added to the private pool of server threads associated with a door during the next door_return() (that has been issued by the current thread after an associated door_bind()). See door_return(3X) . A server thread performing a door_bind() on a door that is already bound to a different door performs an implicit door_unbind() of the previous door.</p> <p>If a process containing threads that have been bound to a door calls <code>fork(2)</code> , the threads in the child process will be bound to an invalid door, and any calls to door_return(3X) will result in an error.</p> |

RETURN VALUES

Upon successful completion, a 0 is returned. Upon failure, a -1 is returned and `errno` is set to indicate the error.

ERRORS

The `door_bind()` and `door_unbind()` functions fail if:

- EBADF** *did* is not a valid door
- EBADF** `door_unbind()` with a server thread that is currently not bound
- EINVAL** *did* was not created with the `DOOR_PRIVATE` attribute

EXAMPLES**EXAMPLE 1** Using `door_bind()`

The following example shows the use of `door_bind()` to create private server pools for two doors, `d1` and `d2`. Function `my_create()` is called when a new server thread is needed; it creates a thread running function, `my_server_create()`, which binds itself to one of the two doors.

```
#include <door.h>
#include <thread.h>
#include <pthread.h>
thread_key_t door_key;
int d1 = -1;
int d2 = -1;
cond_t cv;          /* statically initialized to zero */
mutex_t lock;      /* statically initialized to zero */

extern foo(); extern bar();

static void *
my_server_create(void *arg)
{
    /* wait for d1 & d2 to be initialized */
    mutex_lock(&lock);
    while (d1 == -1 || d2 == -1)
        cond_wait(&cv, &lock);
    mutex_unlock(&lock);

    if (arg == (void *)foo){
        /* bind thread with pool associated with d1 */
        thr_setspecific(door_key, (void *)foo);
        if (door_bind(d1) < 0) {
            perror("door_bind"); exit (-1);
        }
    } else if (arg == (void *)bar) {
        /* bind thread with pool associated with d2 */
        thr_setspecific(door_key, (void *)bar);
        if (door_bind(d2) < 0) {
            /* bind thread to d2 thread pool */
            perror("door_bind"); exit (-1);
        }
    }
    pthread_setcancelstate(POSIX_CANCEL_DISABLE, NULL);
    door_return(NULL, 0, NULL, 0);\011/* Wait for door invocation */
}
```

```

}

static void
my_create(door_info_t *dip)

    /* Pass the door identity information to create function */
    thr_create(NULL, 0, my_server_create, (void *)dip->di_proc,
              THR_BOUND | THR_DETACHED, NULL);
}
main()
{
    (void)door_server_create(my_create);
    mutex_lock(&lock);
    d1 = door_create(foo, NULL, DOOR_PRIVATE); /* Private pool */
    d2 = door_create(bar, NULL, DOOR_PRIVATE); /* Private pool */
    cond_signal(&cv);
    mutex_unlock(&lock);
    while (1)
        pause();
}

```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| Architecture | all |
| Availability | SUNWcsu |
| Stability | Evolving |
| MT-Level | Safe |

SEE ALSO

fork(2), **door_create(3X)**, **door_return(3X)**,
door_server_create(3X), **attributes(5)**

| | |
|--------------------|--|
| NAME | door_call – invoke the function associated with a door descriptor |
| SYNOPSIS | <pre>cc [flag ...] file ... -ldoor [library ...] #include <door.h> typedef struct { char *data_ptr; /* Argument/result buf ptr*/ size_t data_size; /* Argument/result buf size */ door_desc_t *desc_ptr; /* Argument/result descriptors */ uint_t desc_num; /* Argument/result num desc */ char *rbuf; /* Result buffer */ size_t rsize; /* Result buffer size */ } door_arg_t; int door_call(int d, door_arg_t *params);</pre> |
| DESCRIPTION | <p>The door_call() function invokes the function associated with the door descriptor <i>d</i>, and passes the arguments (if any) specified in <i>params</i>. All of the <i>params</i> members are treated as in/out parameters during a door invocation and may be updated upon returning from a door call. Passing NULL for <i>params</i> indicates there are no arguments to be passed and no results expected.</p> <p>Arguments are specified using the <i>data_ptr</i> and <i>desc_ptr</i> members of <i>params</i>. The size of the argument data in bytes is passed in <i>data_size</i> and the number of argument descriptors is passed in <i>desc_num</i>.</p> <p>Results from the door invocation are placed in the buffer, <i>rbuf</i>. See door_return(3X). The <i>data_ptr</i> and <i>desc_ptr</i> members of <i>params</i> are updated to reflect the location of the results within the <i>rbuf</i> buffer. The size of the data results and number of descriptors returned are updated in the <i>data_size</i> and <i>desc_num</i> members. It is acceptable to use the same buffer for input argument data and results, so door_call() may be called with <i>data_ptr</i> and <i>desc_ptr</i> pointing to the buffer <i>rbuf</i>.</p> <p>If the results of a door invocation exceed the size of the buffer specified by <i>rsize</i>, the system automatically allocates a new buffer in the caller's address space and updates the <i>rbuf</i> and <i>rsize</i> members to reflect this location. In this case, the caller is responsible for reclaiming this area using <code>munmap(rbuf, rsize)</code> when the buffer is no longer required. See munmap(2).</p> <p>Descriptors passed in a <i>door_desc_t</i> structure are identified by the <i>d_attributes</i> member. The client marks the <i>d_attributes</i> member with the type of object being passed by logically OR-ing the value of object type. Currently, the only object type that may be passed or returned is a file descriptor, denoted by the DOOR_DESCRIPTOR attribute. Additionally, the DOOR_RELEASE attribute may be set, which will cause the descriptor to be closed in the caller's address space after it is passed to the target. The</p> |

descriptor will be closed even if **door_call()** returns an error, unless that error is EFAULT or EBADF.

The `door_desc_t` structure includes the following members:

```
typedef struct {
    door_attr_t d_attributes; /* Describes the parameter */
    union {
        struct {
            int d_descriptor; /* Descriptor */
            door_id_t d_id; /* Unique door id */
        } d_desc;
    } d_data;
} door_desc_t;
```

When file descriptors are passed or returned, a new descriptor is created in the target address space and the `d_descriptor` member in the target argument is updated to reflect the new descriptor. In addition, the system passes a system-wide unique number associated with each door in the `door_id` member and marks the `d_attributes` member with other attributes associated with a door including the following:

- DOOR_LOCAL The door received was created by this process using **door_create()**. See **door_create(3X)**.
- DOOR_PRIVATE The door received has a private pool of server threads associated with the door.
- DOOR_UNREF The door received is expecting an unreferenced notification.
- DOOR_REVOKED The door received has been revoked by the server.

The **door_call()** function is not a restartable system call. It returns EINTR if a signal was caught and handled by this thread. If the door invocation is not idempotent the caller should mask any signals that may be generated during a **door_call()** operation. If the client aborts in the middle of a **door_call()**, the server thread is notified using the POSIX (see **standards(5)**) thread cancellation mechanism. See **cancellation(3T)**.

The descriptor returned from **door_create()** is marked as close on exec (FD_CLOEXEC). Information about a door is available for all clients of a door using **door_info()**. Programs concerned with security should not place secure information in door data that is accessible by **door_info()**. In particular, secure data should not be stored in the data item *cookie*. See **door_info(3X)**.

RETURN VALUES

Upon successful completion, 0 is returned. Upon failure, -1 is returned and `errno` is set to indicate the error.

ERRORS

The **door_call()** function fails if:

- EBADF** Invalid door descriptor was passed

| | |
|------------------|---|
| EINVAL | Bad arguments were passed |
| EFAULT | Argument pointers pointed outside the allocated address space |
| E2BIG | Arguments were too big for server thread stack |
| EOVERFLOW | System could not create overflow area in caller for results. |
| EAGAIN | Server was out of available resources |
| EINTR | Signal was caught in the client during the invocation |
| EMFILE | The client or server has too many open descriptors |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| Architecture | all |
| Availability | SUNWcsu |
| Stability | Evolving |
| MT-Level | Safe |

SEE ALSO

munmap(2), **cancellation(3T)**, **door_create(3X)**, **door_info(3X)**, **door_return(3X)**, **attributes(5)**, **standards(5)**

| | |
|----------------------|--|
| NAME | door_create – create a door descriptor |
| SYNOPSIS | <pre>cc [flag ...] file ... -ldoor -lthread [library ...] #include <door.h></pre> <p>int door_create(void (*<i>server_procedure</i>) (void *<i>cookie</i>, char *<i>argp</i>, size_t <i>arg_size</i>, door_desc_t *<i>dp</i>, uint_t <i>n_desc</i>, void *<i>cookie</i>, uint_t <i>attributes</i>);</p> |
| DESCRIPTION | <p>The door_create() function creates a door descriptor that describes the procedure specified by the function <i>server_procedure</i>. The data item, <i>cookie</i>, is associated with the door descriptor, and is passed as an argument to the invoked function <i>server_procedure</i> during door_call(3X) invocations. Other arguments passed to <i>server_procedure</i> from an associated door_call() are placed on the stack and include <i>argp</i> and <i>dp</i>. <i>argp</i> points to <i>arg_size</i> bytes of data and <i>dp</i> points to <i>n_desc</i> door_desc_t structures. The <i>attributes</i> flag specifies attributes associated with the newly created door. Valid values for <i>attributes</i> are constructed by OR-ing in one or more of the following values:</p> <p>DOOR_UNREF Delivers a special invocation on the door when the number of descriptors that refer to this door drops to one. In order to trigger this condition, more than one descriptor must have referred to this door at some time. DOOR_UNREF_DATA designates an unreferenced invocation, as the <i>argp</i> argument passed to <i>server_procedure</i>. In the case of an unreferenced invocation, the values for <i>arg_size</i>, <i>dp</i> and <i>n_desc</i> are 0. Only one unreferenced invocation is delivered on behalf of a door.</p> <p>DOOR_PRIVATE Maintains a separate pool of server threads on behalf of the door. Server threads are associated with a door's private server pool using door_bind(3X).</p> <p>The descriptor returned from door_create() will be marked as close on exec (FD_CLOEXEC). Information about a door is available for all clients of a door using door_info(3X). Programs concerned with security should not place secure information in door data that is accessible by door_info(). In particular, secure data should not be stored in the data item <i>cookie</i>.</p> <p>By default, additional threads are created as needed to handle concurrent door_call(3X) invocations. See door_server_create(3X) for information on how to change this behavior.</p> |
| RETURN VALUES | Upon successful completion, door_create() returns a non-negative value. Upon failure, door_create returns -1 and sets <i>errno</i> to indicate the error. |
| ERRORS | <p>The door_create() function fails if:</p> <p>EINVAL Invalid attributes are passed.</p> |

EMFILE The process has too many open descriptors.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| Architecture | all |
| Availability | SUNWcsu |
| Stability | Evolving |
| MT-Level | Safe |

SEE ALSO

door_bind(3X), **door_call(3X)**, **door_info(3X)**, **door_revoke(3X)**,
door_server_create(3X), **attributes(5)**

| NAME | door_cred – return credential information associated with the client | | | | | | | | | | |
|----------------------|--|----------------|-----------------|--------------|-----|--------------|---------|-----------|----------|----------|------|
| SYNOPSIS | <pre>cc [flag ...] file ... -ldoor -lthread [library ...] #include <door.h> int door_cred(door_cred_t *info);</pre> | | | | | | | | | | |
| DESCRIPTION | <p>The door_cred() function returns credential information associated with the client (if any) of the current door invocation.</p> <p>The contents of the <i>info</i> argument include the following fields:</p> <pre>uid_t dc_euid; /* Effective uid of client */ gid_t dc_egid; /* Effective gid of client */ uid_t dc_ruid; /* Real uid of client */ gid_t dc_rgid; /* Real gid of client */ pid_t dc_pid; /* pid of client */</pre> <p>The credential information associated with the client refers to the information from the immediate caller; not necessarily from the first thread in a chain of door calls.</p> | | | | | | | | | | |
| RETURN VALUES | Upon successful completion, door_cred() returns 0. Upon failure, door_cred() returns -1 and sets <code>errno</code> to indicate the error. | | | | | | | | | | |
| ERRORS | <p>The door_cred() function fails if:</p> <p>EFAULT The address of the <i>info</i> argument is invalid.</p> <p>EINVAL There is no associated door client.</p> | | | | | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | | | | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>Architecture</td> <td>all</td> </tr> <tr> <td>Availability</td> <td>SUNWcsu</td> </tr> <tr> <td>Stability</td> <td>Evolving</td> </tr> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | Architecture | all | Availability | SUNWcsu | Stability | Evolving | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | | | | | |
| Architecture | all | | | | | | | | | | |
| Availability | SUNWcsu | | | | | | | | | | |
| Stability | Evolving | | | | | | | | | | |
| MT-Level | Safe | | | | | | | | | | |
| SEE ALSO | door_call(3X) , door_create(3X) , attributes(5) | | | | | | | | | | |

| | |
|----------------------|--|
| NAME | door_info – return information associated with a door descriptor |
| SYNOPSIS | <pre>cc [flag ...] file ... -ldoor [library ...] #include <door.h> int door_info(int d, struct door_info *info);</pre> |
| DESCRIPTION | <p>The door_info() function returns information associated with a door descriptor. door_info() obtains information about the door descriptor <i>d</i> and places the information that is relevant to the door in the structure pointed to by the argument <i>info</i>.</p> <p>The contents of the <i>info</i> argument contains the following fields:</p> <pre>pid_t di_target; /* door server pid */ door_ptr_t di_proc; /* server function */ door_ptr_t di_data; /* data cookie for invocation */ door_attr_t di_attributes; /* door attributes */ door_id_t di_uniquifier; /* unique id among all doors */</pre> <p><i>di_target</i> is the process ID of the door server, or <code>-1</code> if the door server process has exited.</p> <p>The values for <i>di_attributes</i> may be composed of the following:</p> <p>DOOR_LOCAL The door descriptor refers to a service procedure in this process.</p> <p>DOOR_UNREF The door has requested notification when all but the last reference has gone away.</p> <p>DOOR_REVOKED The door descriptor refers to a door that has been revoked.</p> <p>DOOR_PRIVATE The door has a separate pool of server threads associated with it.</p> <p>The <i>di_proc</i> and <i>di_data</i> fields are returned as <code>door_ptr_t</code> objects rather than <code>void *</code> pointers in order to allow clients and servers to interoperate in environments where the pointer sizes may vary in size (for example, 32-bit clients and 64-bit servers). Each door has a system-wide unique number associated with it that is set when the door is created by door_create(). This number is returned in <i>di_uniquifier</i>.</p> |
| RETURN VALUES | Upon successful completion, <code>0</code> is returned. Upon failure, <code>-1</code> is returned and <code>errno</code> is set to indicate the error. |
| ERRORS | <p>The door_info() function fails if:</p> <p>EFAULT The address of argument <i>info</i> is an invalid address.</p> <p>EBADF <i>d</i> is not a door descriptor.</p> |

SEE ALSO | `door_bind(3X)`, `door_create(3X)`, `door_server_create(3X)`

| | |
|----------------------|--|
| NAME | door_return - return from a door invocation |
| SYNOPSIS | <pre>cc [flag ...] file ... -ldoor -lthread [library ...] #include <door.h> int door_return(char *data_ptr, size_t data_size, door_desc_t *desc_ptr, uint_t num_desc);</pre> |
| DESCRIPTION | The door_return() function returns from a door invocation. It returns control to the thread that issued the associated door_call() and blocks waiting for the next door invocation. See door_call(3X) . Results, if any, from the door invocation are passed back to the client in the buffers pointed to by <i>data_ptr</i> and <i>desc_ptr</i> . If there is not a client associated with the door_return() , the calling thread discards the results and blocks waiting for the next door invocation. |
| RETURN VALUES | Upon successful completion, door_return() does not return to the calling process. Upon failure, door_return() returns -1 to the calling process and sets <i>errno</i> to indicate the error. |
| ERRORS | The door_return() function fails and returns to the calling process if: EINVAL Invalid door_return() arguments were passed. EINVAL Thread is bound to a door that no longer exists. EFAULT The address of <i>data_ptr</i> or <i>desc_ptr</i> is invalid. |
| SEE ALSO | door_call(3X) |

| NAME | door_revoke - revoke access to a door descriptor | | | | | | | | | | |
|----------------------|---|----------------|-----------------|--------------|-----|--------------|---------|-----------|----------|----------|------|
| SYNOPSIS | <pre>cc [flag ...] file ... -ldoor -lthread [library ...] #include <door.h> int door_revoke(int d);</pre> | | | | | | | | | | |
| DESCRIPTION | <p>The door_revoke() function revokes access to a door descriptor. Door descriptors are created with door_create(3X). door_revoke() performs an implicit call to close(2), marking the door descriptor <i>d</i> as invalid.</p> <p>A door descriptor can only be revoked by the process that created it. Door invocations that are in progress during a door_revoke() invocation are allowed to complete normally.</p> | | | | | | | | | | |
| RETURN VALUES | Upon successful completion, door_revoke() returns 0. Upon failure, door_revoke() returns -1 and sets <code>errno</code> to indicate the error. | | | | | | | | | | |
| ERRORS | <p>The door_revoke() function fails if:</p> <p>EBADF An invalid door descriptor was passed.</p> <p>EPERM The door descriptor was not created by this process (with door_create(3X)).</p> | | | | | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | | | | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>Architecture</td> <td>all</td> </tr> <tr> <td>Availability</td> <td>SUNWcsu</td> </tr> <tr> <td>Stability</td> <td>Evolving</td> </tr> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | Architecture | all | Availability | SUNWcsu | Stability | Evolving | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | | | | | |
| Architecture | all | | | | | | | | | | |
| Availability | SUNWcsu | | | | | | | | | | |
| Stability | Evolving | | | | | | | | | | |
| MT-Level | Safe | | | | | | | | | | |
| SEE ALSO | close(2) , door_create(3X) , attributes(5) | | | | | | | | | | |

| | |
|--------------------|--|
| NAME | door_server_create – specify an alternative door server thread creation function |
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -ldoor -lthread [<i>library ...</i>] #include <door.h> void (*) () door_server_create(void (*<i>create_proc</i>)(door_info_t*));</pre> |
| DESCRIPTION | <p>Normally, the doors library creates new door server threads in response to incoming concurrent door invocations automatically. There is no pre-defined upper limit on the number of server threads that the system creates in response to incoming invocations (1 server thread for each active door invocation). These threads are created with the default thread stack size and POSIX (see standards(5)) threads cancellation disabled. The created threads also have the <code>THR_BOUND</code> <code>THR_DETACHED</code> attributes for Solaris threads and the <code>PTHREAD_SCOPE_SYSTEM</code> <code>PTHREAD_CREATE_DETACHED</code> attributes for POSIX threads. The signal disposition, and scheduling class of the newly created thread are inherited from the calling thread (initially from the thread calling door_create(), and subsequently from the current active door server thread).</p> <p>The door_server_create() function allows control over the creation of server threads needed for door invocations. The procedure <i>create_proc</i> is called every time the available server thread pool is depleted. In the case of private server pools associated with a door (see the <code>DOOR_PRIVATE</code> attribute in door_create()), information on which pool is depleted is passed to the create function in the form of a <code>door_info_t</code> structure. The <code>di_proc</code> and <code>di_data</code> members of the <code>door_info_t</code> structure may be used as a door identifier associated with the depleted pool. The <i>create_proc</i> procedure may limit the number of server threads created and may also create server threads with appropriate attributes (stack size, thread-specific data, POSIX thread cancellation, signal mask, scheduling attributes, and so forth) for use with door invocations.</p> <p>The specified server creation function should create user level threads using thr_create() with the <code>THR_BOUND</code> flag, or in the case of POSIX threads, pthread_create() with the <code>PTHREAD_SCOPE_SYSTEM</code> attribute. The server threads make themselves available for incoming door invocations on this process by issuing a <code>door_return(NULL, 0, NULL, 0)</code>. In this case, the door_return() arguments are ignored. See door_return(3X) and thr_create(3T).</p> <p>The server threads created by default are enabled for POSIX thread cancellations which may lead to unexpected thread terminations while holding resources (such as locks) if the client aborts the associated door_call(). See door_call(3X). Unless the server code is truly interested in notifications of client aborts during a door invocation and is prepared to handle such notifications using cancellation handlers, POSIX thread cancellation should be</p> |

disabled for server threads using `pthread_setcancelstate` (`PTHREAD_CANCEL_DISABLE`, `NULL`).

The *create_proc* procedure need not create any additional server threads if there is at least one server thread currently active in the process (perhaps handling another door invocation) or it may create as many as seen fit each time it is called. If there are no available server threads during an incoming door invocation, the associated **door_call()** blocks until a server thread becomes available. The *create_proc* procedure must be MT-Safe.

RETURN VALUES

Upon successful completion, **door_server_create()** returns a pointer to the previous server creation function. This function has no failure mode (it cannot fail).

EXAMPLES

EXAMPLE 1 Creating door server threads.

The following example creates door server threads with cancellation disabled and an 8k stack instead of the default stack size:

```
#include <door.h>
#include <pthread.h>
#include <thread.h>

void *
my_thread(void *arg)
{
    pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, NULL);
    door_return(NULL, 0, NULL, 0);
}
void
my_create(door_info_t *dip)
{
    thr_create(NULL, 8192, my_thread, NULL, THR_BOUND | THR_DETACHED, NULL);
}
main()
{
    (void)door_server_create(my_create);
    ...
}
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| Architecture | all |
| Availability | SUNWcsu |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| Stability | Evolving |
| MT-Level | Safe |

SEE ALSO

`cancellation(3T)`, `door_bind(3X)`, `door_call(3X)`, `door_create(3X)`,
`door_return(3X)`, `pthread_create(3T)`,
`pthread_setcancelstate(3T)`, `thr_create(3T)`, `attributes(5)`,
`standards(5)`

| | |
|--------------------|--|
| NAME | drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48 – generate uniformly distributed pseudo-random numbers |
| SYNOPSIS | <pre>#include <stdlib.h> double drand48(void); double erand48(unsigned short x_i [3]); long lrand48(void); long nrand48(unsigned short x_i [3]); long mrand48(void); long jrand48(unsigned short x_i [3]); void srand48(long seedval); unsigned short * seed48(unsigned short seed16v[3]); void lcong48(unsigned short param[7]);</pre> |
| DESCRIPTION | <p>This family of functions generates pseudo-random numbers using the well-known linear congruential algorithm and 48-bit integer arithmetic.</p> <p>Functions drand48() and erand48() return non-negative double-precision floating-point values uniformly distributed over the interval [0.0, 1.0].</p> <p>Functions lrand48() and nrand48() return non-negative long integers uniformly distributed over the interval [0, 2³¹].</p> <p>Functions mrnd48() and jrnd48() return signed long integers uniformly distributed over the interval [-2³¹, 2³¹].</p> <p>Functions srand48(), seed48(), and lcong48() are initialization entry points, one of which should be invoked before either drand48(), lrand48(), or mrnd48() is called. (Although it is not recommended practice, constant default initializer values will be supplied automatically if drand48(), lrand48(), or mrnd48() is called without a prior call to an initialization entry point.) Functions erand48(), nrand48(), and jrnd48() do not require an initialization entry point to be called first.</p> <p>All the routines work by generating a sequence of 48-bit integer values, X_i, according to the linear congruential formula</p> $X_{n+1} = (aX_n + c) \bmod m \quad n \geq 0.$ <p>The parameter $m = 2^{48}$; hence 48-bit integer arithmetic is performed. Unless lcong48() has been invoked, the multiplier value a and the addend value c are given by</p> |

$$a = 5DEECE66D_{16} = 273673163155_8$$

$$c = B_{16} = 13_8.$$

The value returned by any of the functions **drand48()**, **erand48()**, **lrand48()**, **nrand48()**, **rand48()**, or **jrand48()** is computed by first generating the next 48-bit X_i in the sequence. Then the appropriate number of bits, according to the type of data item to be returned, are copied from the high-order (leftmost) bits of X_i and transformed into the returned value.

The functions **drand48()**, **lrand48()**, and **rand48()** store the last 48-bit X_i generated in an internal buffer. X_i must be initialized prior to being invoked. The functions **erand48()**, **nrand48()**, and **jrand48()** require the calling program to provide storage for the successive X_i values in the array specified as an argument when the functions are invoked. These routines do not have to be initialized; the calling program must place the desired initial value of X_i into the array and pass it as an argument. By using different arguments, functions **erand48()**, **nrand48()**, and **jrand48()** allow separate modules of a large program to generate several *independent* streams of pseudo-random numbers, that is, the sequence of numbers in each stream will *not* depend upon how many times the routines have been called to generate numbers for the other streams.

The initializer function **srand48()** sets the high-order 32 bits of X_i to the 32 bits contained in its argument. The low-order 16 bits of X_i are set to the arbitrary value $330E_{16}$.

The initializer function **seed48()** sets the value of X_i to the 48-bit value specified in the argument array. In addition, the previous value of X_i is copied into a 48-bit internal buffer, used only by **seed48()**, and a pointer to this buffer is the value returned by **seed48()**. This returned pointer, which can just be ignored if not needed, is useful if a program is to be restarted from a given point at some future time — use the pointer to get at and store the last X_i value, and then use this value to reinitialize using **seed48()** when the program is restarted.

The initialization function **lcong48()** allows the user to specify the initial X_i , the multiplier value a , and the addend value c . Argument array elements *param[0-2]* specify X_i , *param[3-5]* specify the multiplier a , and *param[6]* specifies the 16-bit addend c . After **lcong48()** has been called, a subsequent call to either **srand48()** or **seed48()** will restore the “standard” multiplier and addend values, a and c , specified above.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO `rand(3C)` , `attributes(5)`

NAME dup2 – duplicate an open file descriptor

SYNOPSIS #include <unistd.h>

```
int dup2(int fdes, int fdes2);
```

DESCRIPTION The **dup2()** function causes the file descriptor *fdes2* to refer to the same file as *fdes*. The *fdes* argument is a file descriptor referring to an open file, and *fdes2* is a non-negative integer less than the current value for the maximum number of open file descriptors allowed the calling process. See **getrlimit(2)**. If *fdes2* already refers to an open file, not *fdes*, it is closed first. If *fdes2* refers to *fdes*, or if *fdes* is not a valid open file descriptor, *fdes2* will not be closed first.

The **dup2()** function is equivalent to `fcntl(fdes, F_DUP2FD, fdes2)`.

RETURN VALUES Upon successful completion a non-negative integer representing the file descriptor is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.

ERRORS The **dup2()** function will fail if:

EBADF The *fdes* argument is not a valid open file descriptor.

EBADF The *fdes2* argument is negative or is not less than the current resource limit returned by `getrlimit(RLIMIT_NOFILE, ...)`.

EINTR A signal was caught during the **dup2()** call.

EMFILE The process has too many open files. See **fcntl(2)**.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO **close(2)**, **creat(2)**, **exec(2)**, **fcntl(2)**, **getrlimit(2)**, **open(2)**, **pipe(2)**, **lockf(3C)**, **attributes(5)**

| | |
|----------------------|--|
| NAME | dupwin – duplicate a window |
| SYNOPSIS | <pre>#include <curses.h> WINDOW *dupwin(WINDOW *win);</pre> |
| PARAMETERS | <p>win Is a pointer to the window that is to be duplicated.</p> |
| DESCRIPTION | The dupwin() function creates a duplicate of window <i>win</i> . A pointer to the new window structure is returned. |
| RETURN VALUES | On success, this function returns a pointer to new window structure; otherwise, it returns a null pointer. |
| ERRORS | None. |
| SEE ALSO | delwin(3XC) , derwin(3XC) |

| | |
|----------------------|---|
| NAME | echo, noecho – enable/disable terminal echo |
| SYNOPSIS | <pre>#include <curses.h> int echo(void); int noecho(void);</pre> |
| DESCRIPTION | <p>The echo() and noecho() functions enable and disable terminal echo, respectively. When enabled, characters received by getch(3XC) are echoed back to the terminal. When disabled, characters are transferred to the program without echoing them to the terminal display. The program may instead echo the characters to an area of the screen controlled by the program or may not echo the characters at all. Terminal echo is enabled, by default.</p> <p>Subsequent calls to echo() or noecho() do not flush type-ahead.</p> <p>The tty driver echo is disabled by initscr(3XC) and newterm(3XC). All echoing is controlled by X/Open Curses.</p> |
| RETURN VALUES | On success, these functions return OK . Otherwise, they return ERR . |
| ERRORS | None. |
| SEE ALSO | getch(3XC) , getstr(3XC) , initscr(3XC) , scanw(3XC) |

| | | | | | |
|----------------------|--|------------------|--|-------------------|---|
| NAME | echochar, wechochar – add a single-byte character and refresh window | | | | |
| SYNOPSIS | <pre>#include <curses.h> int echochar(const chtype <i>ch</i>); int wechochar(WINDOW * <i>win</i>, const chtype <i>ch</i>);</pre> | | | | |
| PARAMETERS | <table><tr><td><i>ch</i></td><td>Is a pointer to the character to be written to the window.</td></tr><tr><td><i>win</i></td><td>Is a pointer to the window in which the character is to be added.</td></tr></table> | <i>ch</i> | Is a pointer to the character to be written to the window. | <i>win</i> | Is a pointer to the window in which the character is to be added. |
| <i>ch</i> | Is a pointer to the character to be written to the window. | | | | |
| <i>win</i> | Is a pointer to the window in which the character is to be added. | | | | |
| DESCRIPTION | The echochar() function produces the same effect as calling addch(3XC) and then refresh(3XC) . The wechochar() function produces the same effect as calling waddch(3XC) and then wrefresh(3XC) . | | | | |
| RETURN VALUES | On success, these functions return OK . Otherwise, they return ERR . | | | | |
| ERRORS | None. | | | | |
| SEE ALSO | addch(3XC) , doupdate(3XC) , echo_wchar(3XC) | | | | |

| | | | | | |
|----------------------|--|------------|--|------------|---|
| NAME | echo_wchar, wecho_wchar – add a complex character and refresh window | | | | |
| SYNOPSIS | <pre>#include <curses.h> int echo_wchar(const cchar_t * wch); int wecho_wchar(WINDOW * win, const cchar_t * wch);</pre> | | | | |
| PARAMETERS | <table><tr><td>wch</td><td>Is a pointer to the complex character to be written to the window.</td></tr><tr><td>win</td><td>Is a pointer to the window in which the character is to be added.</td></tr></table> | wch | Is a pointer to the complex character to be written to the window. | win | Is a pointer to the window in which the character is to be added. |
| wch | Is a pointer to the complex character to be written to the window. | | | | |
| win | Is a pointer to the window in which the character is to be added. | | | | |
| DESCRIPTION | The echo_wchar() function produces the same effect as calling add_wch(3XC) and then refresh(3XC) . The wecho_wchar() function produces the same effect as calling wadd_wch(3XC) and then wrefresh(3XC) . | | | | |
| RETURN VALUES | On success, these functions return OK . Otherwise, they return ERR . | | | | |
| ERRORS | None. | | | | |
| SEE ALSO | add_wch(3XC) , doupdate(3XC) , echochar(3XC) | | | | |

| | |
|--------------------|---|
| NAME | econvert, fconvert, gconvert, seconvert, sfconvert, sgconvert, qeconvert, qfconvert, qgconvert, ecvt, fcvt, gcvt – output conversion |
| SYNOPSIS | <pre>#include <floatingpoint.h> char * econvert(double value, int ndigit, int * decpt, int * sign, char * buf); char * fconvert(double value, int ndigit, int * decpt, int * sign, char * buf); char * gconvert(double value, int ndigit, int trailing, char * buf); char * seconvert(single * value, int ndigit, int * decpt, int * sign, char * buf); char * sfconvert(single * value, int ndigit, int * decpt, int * sign, char * buf); char * sgconvert(single * value, int ndigit, int trailing, char * buf); char * qeconvert(quadruple * value, int ndigit, int * decpt, int * sign, char * buf); char * qfconvert(quadruple * value, int ndigit, int * decpt, int * sign, char * buf); char * qgconvert(quadruple * value, int ndigit, int trailing, char * buf); char * ecvt(double value, int ndigit, int * decpt, int * sign); char * fcvt(double value, int ndigit, int * decpt, int * sign); char * gcvt(double value, int ndigit, char * buf);</pre> |
| DESCRIPTION | <p>The econvert() function converts the <i>value</i> to a null-terminated string of <i>ndigit</i> ASCII digits in <i>buf</i> and returns a pointer to <i>buf</i>. <i>buf</i> should contain at least <i>ndigit+1</i> characters. The position of the decimal point relative to the beginning of the string is stored indirectly through <i>decpt</i>. Thus <i>buf</i> == "314" and <i>*decpt</i> == 1 corresponds to the numerical value 3.14, while <i>buf</i> == "314" and <i>*decpt</i> == -1 corresponds to the numerical value .0314. If the sign of the result is negative, the word pointed to by <i>sign</i> is nonzero; otherwise it is zero. The least significant digit is rounded.</p> <p>The fconvert() function works much like econvert(), except that the correct digit has been rounded as if for <code>sprintf(%w.nf)</code> output with $n = ndigit$ digits to the right of the decimal point. <i>ndigit</i> can be negative to indicate rounding to the left of the decimal point. The return value is a pointer to <i>buf</i>. <i>buf</i> should contain at least $310 + \max(0, ndigit)$ characters to accommodate any double-precision <i>value</i>.</p> <p>The gconvert() function converts the <i>value</i> to a null-terminated ASCII string in <i>buf</i> and returns a pointer to <i>buf</i>. It produces <i>ndigit</i> significant digits in fixed-decimal format, like <code>sprintf(%w.nf)</code>, if possible, and otherwise in floating-decimal format, like <code>sprintf(%w.ne)</code>; in either case <i>buf</i> is</p> |

```
(void) sprintf(buf, ``%w.ng``,value) ;
```

ready for printing, with sign and exponent. The result corresponds to that obtained by

If *trailing* = 0, trailing zeros and a trailing point are suppressed, as in `sprintf(%g)`. If *trailing* != 0, trailing zeros and a trailing point are retained, as in `sprintf(%#g)`.

The **seconvert()**, **sfconvert()**, and **sgconvert()** functions are single-precision versions of these functions, and are more efficient than the corresponding double-precision versions. A pointer rather than the value itself is passed to avoid C's usual conversion of single-precision arguments to double.

The **qeconvert()**, **qfconvert()**, and **qgconvert()** functions are quadruple-precision versions of these functions. The **qfconvert()** function can overflow the *decimal_record* field *ds* if *value* is too large. In that case, *buf[0]* is set to zero.

The **ecvt()** and **fcvt()** functions are versions of **econvert()** and **fconvert()** that create a string in a static data area, overwritten by each call, and return values that point to that static data. These functions are therefore not reentrant.

The **gcvt()** function is a version of **gconvert()** that always suppresses trailing zeros and point.

IEEE Infinities and NaNs are treated similarly by these functions. "NaN" is returned for NaN, and "Inf" or "Infinity" for Infinity. The longer form is produced when *ndigit* >= 8.

ATTRIBUTES

See **attributes** (5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

sprintf(3S), **attributes(5)**

| | |
|----------------------|--|
| NAME | ecvt, fcvt, gcvt – convert floating-point number to string |
| SYNOPSIS | <pre>#include <stdlib.h> char * ecvt(double value, int ndigit, int * decpt, int * sign); char * fcvt(double value, int ndigit, int * decpt, int * sign); char * gcvt(double value, int ndigit, char * buf);</pre> |
| DESCRIPTION | <p>The ecvt() , fcvt() and gcvt() functions convert floating-point numbers to null-terminated strings.</p> <p>ecvt() The ecvt() function converts <i>value</i> to a null-terminated string of <i>ndigit</i> digits (where <i>ndigit</i> is reduced to an unspecified limit determined by the precision of a <code>double</code>)and returns a pointer to the string. The high-order digit is non-zero, unless the value is 0. The low-order digit is rounded. The position of the radix character relative to the beginning of the string is stored in the integer pointed to by <i>decpt</i> (negative means to the left of the returned digits). The radix character is not included in the returned string. If the sign of the result is negative, the integer pointed to by <i>sign</i> is non-zero, otherwise it is 0.</p> <p>If the converted value is out of range or is not representable, the contents of the returned string are unspecified.</p> <p>fcvt() The fcvt() function is identical to ecvt() except that <i>ndigit</i> specifies the number of digits desired after the radix point. The total number of digits in the result string is restricted to an unspecified limit as determined by the precision of a <code>double</code> .</p> <p>gcvt() The gcvt() function converts <i>value</i> to a null-terminated string (similar to that of the <code>%g</code> format of <code>printf(3S)</code>)in the array pointed to by <i>buf</i> and returns <i>buf</i> . It produces <i>ndigit</i> significant digits (limited to an unspecified value determined by the precision of a <code>double</code>)in <code>%f</code> if possible, or <code>%e</code> (scientific notation) otherwise. A minus sign is included in the returned string if <i>value</i> is less than 0. A radix character is included in the returned string if <i>value</i> is not a whole number. Trailing zeros are suppressed where <i>value</i> is not a whole number. The radix character is determined by the current locale. If <code>setlocale(3C)</code> has not been called successfully, the default locale, POSIX, is used. The default locale specifies a period (<code>.</code>)as the radix character. The <code>LC_NUMERIC</code> category determines the value of the radix character within the current locale.</p> |
| RETURN VALUES | <p>The ecvt() and fcvt() functions return a pointer to a null-terminated string of digits.</p> <p>The gcvt() function returns <i>buf</i> .</p> |

ERRORS No errors are defined.

USAGE The return values from `ecvt()` and `fcvt()` may point to static data which may be overwritten by subsequent calls to these functions.

For portability to implementations conforming to earlier versions of this document, `sprintf(3S)` is preferred over this function.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `printf(3S)`, `setlocale(3C)`, `sprintf(3S)`, `attributes(5)`

| NAME | elf32_fsize, elf64_fsize – return the size of an object file type | | | | |
|--------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lelf [library ...] #include <libelf.h> size_t elf32_fsize(Elf_Type type, size_t count, unsigned ver); size_t elf64_fsize(Elf_Type type, size_t count, unsigned ver);</pre> | | | | |
| DESCRIPTION | <p>elf32_fsize() gives the size in bytes of the 32-bit file representation of <i>count</i> data objects with the given <i>type</i>. The library uses version <i>ver</i> to calculate the size. See elf(3E) and elf_version(3E).</p> <p>Constant values are available for the sizes of fundamental types:</p> <pre>Elf_Type\011File Size\011Memory Size ELF_T_ADDR\011ELF32_FSZ_ADDR\011sizeof(Elf32_Addr) ELF_T_BYTE\011\011sizeof(unsigned char) ELF_T_HALF\011ELF32_FSZ_HALF\011sizeof(Elf32_Half) ELF_T_OFF\011ELF32_FSZ_OFF\011sizeof(Elf32_Off) ELF_T_SWORD\011ELF32_FSZ_SWORD\011sizeof(Elf32_Sword) ELF_T_WORD\011ELF32_FSZ_WORD\011sizeof(Elf32_Word)</pre> <p>elf32_fsize() returns 0 if the value of <i>type</i> or <i>ver</i> is unknown. See elf32_xlatetof(3E) for a list of the <i>type</i> values.</p> <p>For the 64-bit class, replace 32 with 64 as appropriate.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | elf(3E) , elf32_xlatetof(3E) , elf_version(3E) , attributes(5) | | | | |

| | |
|--------------------|--|
| NAME | elf32_getehdr, elf32_newehdr, elf64_getehdr, elf64_newehdr – retrieve class-dependent object file header |
| SYNOPSIS | <pre> cc [flag ...] file ... -lelf [library ...] #include <libelf.h> Elf32_Ehdr * elf32_getehdr(Elf * elf); Elf32_Ehdr * elf32_newehdr(Elf * elf); Elf64_Ehdr * elf64_getehdr(Elf * elf); Elf64_Ehdr * elf64_newehdr(Elf * elf); </pre> |
| DESCRIPTION | <p>For a 32-bit class file, elf32_getehdr() returns a pointer to an ELF header, if one is available for the ELF descriptor <i>elf</i>. If no header exists for the descriptor, elf32_newehdr() allocates a clean one, but it otherwise behaves the same as elf32_getehdr(). It does not allocate a new header if one exists already. If no header exists for elf32_getehdr(), one cannot be created for elf32_newehdr(), a system error occurs, the file is not a 32-bit class file, or <i>elf</i> is null, both functions return a null pointer.</p> <p>For the 64-bit class, replace 32 with 64 as appropriate.</p> <p>The header includes the following members:</p> <pre> unsigned char\011e_ident[EI_NIDENT]; Elf32_Half\011e_type; Elf32_Half\011e_machine; Elf32_Word\011e_version; Elf32_Addr\011e_entry; Elf32_Off\011e_phoff; Elf32_Off\011e_shoff; Elf32_Word\011e_flags; Elf32_Half\011e_ehsize; Elf32_Half\011e_phentsize; Elf32_Half\011e_phnum; Elf32_Half\011e_shentsize; Elf32_Half\011e_shnum; Elf32_Half\011e_shstrndx; </pre> |

elf32_newehdr() automatically sets the `ELF_F_DIRTY` bit. See **elf_flagdata(3E)**. A program may use **elf_getident()** to inspect the identification bytes from a file.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

elf(3E), **elf_begin(3E)**, **elf_flagdata(3E)**, **elf_getident(3E)**, **attributes(5)**

| | |
|--------------------|--|
| NAME | elf32_getphdr, elf32_newphdr, elf64_getphdr, elf64_newphdr – retrieve class-dependent program header table |
| SYNOPSIS | <pre>cc [flag ...] file ... -lelf [library ...] #include <libelf.h> Elf32_Phdr * elf32_getphdr(Elf * elf); Elf32_Phdr * elf32_newphdr(Elf * elf, size_t count); Elf64_Phdr * elf64_getphdr(Elf * elf); Elf64_Phdr * elf64_newphdr(Elf * elf, size_t count);</pre> |
| DESCRIPTION | <p>For a 32-bit class file, elf32_getphdr() returns a pointer to the program execution header table, if one is available for the ELF descriptor <i>elf</i> .</p> <p>elf32_newphdr() allocates a new table with <i>count</i> entries, regardless of whether one existed previously, and sets the <code>ELF_F_DIRTY</code> bit for the table. See elf_flagdata(3E) . Specifying a zero <i>count</i> deletes an existing table. Note this behavior differs from that of elf32_newehdr() allowing a program to replace or delete the program header table, changing its size if necessary. See elf32_getehdr(3E) .</p> <p>If no program header table exists, the file is not a 32-bit class file, an error occurs, or <i>elf</i> is <code>NULL</code> , both functions return a null pointer. Additionally, elf32_newphdr() returns a null pointer if <i>count</i> is 0 .</p> <p>The table is an array of <code>Elf32_Phdr</code> structures, each of which includes the following members:</p> <pre>Elf32_Word\011p_type; Elf32_Off\011p_offset; Elf32_Addr\011p_vaddr; Elf32_Addr\011p_paddr; Elf32_Word\011p_filesz; Elf32_Word\011p_memsz; Elf32_Word\011p_flags; Elf32_Word\011p_align;</pre> |

The `Elf64_Phdr` structures include the following members:

```
Elf64_Word\011p_type;
Elf64_Word\011p_flags;
Elf64_Off\011p_offset;
Elf64_Addr\011p_vaddr;
Elf64_Addr\011p_paddr;
Elf64_Xword\011p_filesz;
Elf64_Xword\011p_memsz;
Elf64_Xword\011p_align;
```

For the 64-bit class, replace 32 with 64 as appropriate.

The ELF header's `e_phnum` member tells how many entries the program header table has. See `elf32_getehdr(3E)`. A program may inspect this value to determine the size of an existing table; `elf32_newphdr()` automatically sets the member's value to `count`. If the program is building a new file, it is responsible for creating the file's ELF header before creating the program header table.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`elf(3E)`, `elf32_getehdr(3E)`, `elf_begin(3E)`, `elf_flagdata(3E)`, `attributes(5)`

| | |
|--------------------|---|
| NAME | elf32_getshdr, elf64_getshdr – retrieve class-dependent section header |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lelf [<i>library</i> ...] #include <libelf.h> Elf32_Shdr * elf32_getshdr(Elf_Scn * <i>scn</i>); Elf64_Shdr * elf64_getshdr(Elf_Scn * <i>scn</i>);</pre> |
| DESCRIPTION | <p>For a 32-bit class file, elf32_getshdr() returns a pointer to a section header for the section descriptor <i>scn</i> . Otherwise, the file is not a 32-bit class file, <i>scn</i> was NULL , or an error occurred; elf32_getshdr() then returns NULL .</p> <p>The elf32_getshdr header includes the following members:</p> <pre>Elf32_Word\011sh_name; Elf32_Word\011sh_type; Elf32_Word\011sh_flags; Elf32_Addr\011sh_addr; Elf32_Off\011sh_offset; Elf32_Word\011sh_size; Elf32_Word\011sh_link; Elf32_Word\011sh_info; Elf32_Word\011sh_addralign; Elf32_Word\011sh_entsize;</pre> <p>while the elf64_getshdr header includes the following members:</p> <pre>Elf64_Word\011sh_name; Elf64_Word\011sh_type; Elf64_Xword\011sh_flags; Elf64_Addr\011sh_addr; Elf64_Off\011sh_offset; Elf64_Xword\011sh_size; Elf64_Word\011sh_link; Elf64_Word\011sh_info; Elf64_Xword\011sh_addralign; Elf64_Xword\011sh_entsize;</pre> |

For the 64-bit class, replace 32 with 64 as appropriate.

If the program is building a new file, it is responsible for creating the file's ELF header before creating sections.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`elf(3E)`, `elf_flagdata(3E)`, `elf_getscn(3E)`, `elf_strptr(3E)`, `attributes(5)`

| | | | | | | | |
|---------------------|---|--------------------|--|---------------------|---|---------------------|---|
| NAME | elf32_xlatetof, elf32_xlatetom, elf64_xlatetof, elf64_xlatetom – class-dependent data translation | | | | | | |
| SYNOPSIS | <pre>cc [flag ...] file ... -lelf [library ...] #include <libelf.h> Elf_Data * elf32_xlatetof(Elf_Data * dst, const Elf_Data * src, unsigned encode); Elf_Data * elf32_xlatetom(Elf_Data * dst, const Elf_Data * src, unsigned encode); Elf_Data * elf64_xlatetof(Elf_Data * dst, const Elf_Data * src, unsigned encode); Elf_Data * elf64_xlatetom(Elf_Data * dst, const Elf_Data * src, unsigned encode);</pre> | | | | | | |
| DESCRIPTION | <p>elf32_xlatetom() translates various data structures from their 32-bit class file representations to their memory representations; elf32_xlatetof() provides the inverse. This conversion is particularly important for cross development environments. <i>src</i> is a pointer to the source buffer that holds the original data; <i>dst</i> is a pointer to a destination buffer that will hold the translated copy. <i>encode</i> gives the byte encoding in which the file objects are to be represented and must have one of the encoding values defined for the ELF header's <code>e_ident[EI_DATA]</code> entry (see elf_getident(3E)). If the data can be translated, the functions return <i>dst</i>. Otherwise, they return NULL because an error occurred, such as incompatible types, destination buffer overflow, etc.</p> <p>elf_getdata(3E) describes the <code>Elf_Data</code> descriptor, which the translation routines use as follows:</p> <table border="0" style="margin-left: 20px;"> <tr> <td style="padding-right: 20px;"><code>d_buf</code></td> <td>Both the source and destination must have valid buffer pointers.</td> </tr> <tr> <td style="padding-right: 20px;"><code>d_type</code></td> <td>This member's value specifies the type of the data to which <code>d_buf</code> points and the type of data to be created in the destination. The program supplies a <code>d_type</code> value in the source; the library sets the destination's <code>d_type</code> to the same value. These values are summarized below.</td> </tr> <tr> <td style="padding-right: 20px;"><code>d_size</code></td> <td>This member holds the total size, in bytes, of the memory occupied by the source data and the size allocated for the destination data. If the destination buffer is not large enough, the routines do not change its original contents. The</td> </tr> </table> | <code>d_buf</code> | Both the source and destination must have valid buffer pointers. | <code>d_type</code> | This member's value specifies the type of the data to which <code>d_buf</code> points and the type of data to be created in the destination. The program supplies a <code>d_type</code> value in the source; the library sets the destination's <code>d_type</code> to the same value. These values are summarized below. | <code>d_size</code> | This member holds the total size, in bytes, of the memory occupied by the source data and the size allocated for the destination data. If the destination buffer is not large enough, the routines do not change its original contents. The |
| <code>d_buf</code> | Both the source and destination must have valid buffer pointers. | | | | | | |
| <code>d_type</code> | This member's value specifies the type of the data to which <code>d_buf</code> points and the type of data to be created in the destination. The program supplies a <code>d_type</code> value in the source; the library sets the destination's <code>d_type</code> to the same value. These values are summarized below. | | | | | | |
| <code>d_size</code> | This member holds the total size, in bytes, of the memory occupied by the source data and the size allocated for the destination data. If the destination buffer is not large enough, the routines do not change its original contents. The | | | | | | |

translation routines reset the destination's `d_size` member to the actual size required, after the translation occurs. The source and destination sizes may differ.

`d_version` This member holds the version number of the objects (desired) in the buffer. The source and destination versions are independent.

Translation routines allow the source and destination buffers to coincide. That is, `dst→d_buf` may equal `src→d_buf` . Other cases where the source and destination buffers overlap give undefined behavior.

```
Elf_Type    \01132-Bit Memory Type
ELF_T_ADDR\011Elf32_Addr
ELF_T_BYTE\011unsigned char
ELF_T_DYN\011Elf32_Dyn
ELF_T_EHDR\011Elf32_Ehdr
ELF_T_HALF\011Elf32_Half
ELF_T_OFF\011Elf32_Off
ELF_T_PHDR\011Elf32_Phdr
ELF_T_REL\011Elf32_Rel
ELF_T_RELA\011Elf32_Rela
ELF_T_SHDR\011Elf32_Shdr
ELF_T_SWORD\011Elf32_Sword
ELF_T_SYM\011Elf32_Sym
ELF_T_WORD\011Elf32_Word
```

Translating buffers of type `ELF_T_BYTE` does not change the byte order.

For the 64-bit class, replace 32 with 64 as appropriate.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`elf(3E)` , `elf32_fsize(3E)` , `elf_getdata(3E)` , `elf_getident(3E)` , `attributes(5)`

| | |
|---------------------|---|
| NAME | elf – object file access library |
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lelf [<i>library ...</i>] #include <libelf.h></pre> |
| DESCRIPTION | <p>Functions in the ELF access library let a program manipulate ELF (Executable and Linking Format) object files, archive files, and archive members. The header provides type and function declarations for all library services.</p> <p>Programs communicate with many of the higher-level routines using an <i>ELF descriptor</i>. That is, when the program starts working with a file, <code>elf_begin(3E)</code> creates an ELF descriptor through which the program manipulates the structures and information in the file. These ELF descriptors can be used both to read and to write files. After the program establishes an ELF descriptor for a file, it may then obtain <i>section descriptors</i> to manipulate the sections of the file (see <code>elf_getscn(3E)</code>). Sections hold the bulk of an object file's real information, such as text, data, the symbol table, and so on. A section descriptor "belongs" to a particular ELF descriptor, just as a section belongs to a file. Finally, <i>data descriptors</i> are available through section descriptors, allowing the program to manipulate the information associated with a section. A data descriptor "belongs" to a section descriptor.</p> <p>Descriptors provide private handles to a file and its pieces. In other words, a data descriptor is associated with one section descriptor, which is associated with one ELF descriptor, which is associated with one file. Although descriptors are private, they give access to data that may be shared. Consider programs that combine input files, using incoming data to create or update another file. Such a program might get data descriptors for an input and an output section. It then could update the output descriptor to reuse the input descriptor's data. That is, the descriptors are distinct, but they could share the associated data bytes. This sharing avoids the space overhead for duplicate buffers and the performance overhead for copying data unnecessarily.</p> |
| File Classes | <p>ELF provides a framework in which to define a family of object files, supporting multiple processors and architectures. An important distinction among object files is the <i>class</i>, or capacity, of the file. The 32-bit class supports architectures in which a 32-bit object can represent addresses, file sizes, and so on, as in the following:</p> <pre>Name Purpose Elf32_Addr Unsigned address Elf32_Half Unsigned medium integer Elf32_Off Unsigned file offset Elf32_Sword Signed large integer Elf32_Word Unsigned large integer unsigned char Unsigned small integer</pre> |

The 64-bit class works the same as the 32-bit class, substituting 64 for 32 as necessary. Other classes will be defined as necessary, to support larger (or smaller) machines. Some library services deal only with data objects for a specific class, while others are class-independent. To make this distinction clear, library function names reflect their status, as described below.

Data Representation

Conceptually, two parallel sets of objects support cross compilation environments. One set corresponds to file contents, while the other set corresponds to the native memory image of the program manipulating the file. Type definitions supplied by the headers work on the native machine, which may have different data encodings (size, byte order, and so on) than the target machine. Although native memory objects should be at least as big as the file objects (to avoid information loss), they may be bigger if that is more natural for the host machine.

Translation facilities exist to convert between file and memory representations. Some library routines convert data automatically, while others leave conversion as the program's responsibility. Either way, programs that create object files must write file-typed objects to those files; programs that read object files must take a similar view. See `elf32_xlatetof(3E)` and `elf32_fsize(3E)` for more information.

Programs may translate data explicitly, taking full control over the object file layout and semantics. If the program prefers not to have and exercise complete control, the library provides a higher-level interface that hides many object file details. `elf_begin()` and related functions let a program deal with the native memory types, converting between memory objects and their file equivalents automatically when reading or writing an object file.

ELF Versions

Object file versions allow ELF to adapt to new requirements. *Three independent versions* can be important to a program. First, an application program knows about a particular version by virtue of being compiled with certain headers. Second, the access library similarly is compiled with header files that control what versions it understands. Third, an ELF object file holds a value identifying its version, determined by the ELF version known by the file's creator. Ideally, all three versions would be the same, but they may differ. If a program's version is newer than the access library, the program might use information unknown to the library. Translation routines might not work properly, leading to undefined behavior. This condition merits installing a new library.

The library's version might be newer than the program's and the file's. The library understands old versions, thus avoiding compatibility problems in this case.

Finally, a file's version might be newer than either the program or the library understands. The program might or might not be able to process the file properly, depending on whether the file has extra information and whether that information can be safely ignored. Again, the safe alternative is to install a new library that understands the file's version.

To accommodate these differences, a program must use `elf_version(3E)` to pass its version to the library, thus establishing the *working version* for the process. Using this, the library accepts data from and presents data to the program in the proper representations. When the library reads object files, it uses each file's version to interpret the data. When writing files or converting memory types to the file equivalents, the library uses the program's working version for the file data.

System Services

As mentioned above, `elf_begin()` and related routines provide a higher-level interface to ELF files, performing input and output on behalf of the application program. These routines assume a program can hold entire files in memory, without explicitly using temporary files. When reading a file, the library routines bring the data into memory and perform subsequent operations on the memory copy. Programs that wish to read or write large object files with this model must execute on a machine with a large process virtual address space. If the underlying operating system limits the number of open files, a program can use `elf_cntl(3E)` to retrieve all necessary data from the file, allowing the program to close the file descriptor and reuse it.

Although the `elf_begin()` interfaces are convenient and efficient for many programs, they might be inappropriate for some. In those cases, an application may invoke the `elf32_xlatetom(3E)` or `elf32_xlatetof(3E)` data translation routines directly. These routines perform no input or output, leaving that as the application's responsibility. By assuming a larger share of the job, an application controls its input and output model.

Library Names

Names associated with the library take several forms.

| | |
|--|---|
| <code>elf_</code> <i>name</i> | These class-independent names perform some service, <i>name</i> , for the program. |
| <code>elf32_</code> <i>name</i> | Service names with an embedded class, 32 here, indicate they work only for the designated class of files. |
| <code>Elf_</code> <i>Type</i> | Data types can be class-independent as well, distinguished by <i>Type</i> . |
| <code>Elf32_</code> <i>Type</i> | Class-dependent data types have an embedded class name, 32 here. |

| | |
|-----------------------|--|
| ELF_C_CMD | Several functions take commands that control their actions. These values are members of the <code>Elf_Cmd</code> enumeration; they range from zero through <code>ELF_C_NUM-1</code> . |
| ELF_F_FLAG | Several functions take flags that control library status and/or actions. Flags are bits that may be combined. |
| ELF32_FSZ_TYPE | These constants give the file sizes in bytes of the basic ELF types for the 32-bit class of files. See <code>elf32_fsize()</code> for more information. |
| ELF_K_KIND | The function <code>elf_kind()</code> identifies the <i>KIND</i> of file associated with an ELF descriptor. These values are members of the <code>Elf_Kind</code> enumeration; they range from zero through <code>ELF_K_NUM-1</code> . |
| ELF_T_TYPE | When a service function, such as <code>elf32_xlatetom()</code> or <code>elf32_xlatetof()</code> , deals with multiple types, names of this form specify the desired <i>TYPE</i> . Thus, for example, <code>ELF_T_EHDR</code> is directly related to <code>Elf32_Ehdr</code> . These values are members of the <code>Elf_Type</code> enumeration; they range from zero through <code>ELF_T_NUM-1</code> . |

EXAMPLES

EXAMPLE 1 An interpretation of elf file.

The basic interpretation of an ELF file consists of:

- opening an ELF object file
- obtaining an ELF descriptor
- analyzing the file using the descriptor.

The following example opens the file, obtains the ELF descriptor, and prints out the names of each section in the file.

```
#include <fcntl.h>
#include <stdio.h>
#include <libelf.h>
#include <stdlib.h>
#include <string.h>
static void failure(void);
void
main(int argc, char ** argv)
{
    Elf32_Shdr * shdr;
    Elf32_Ehdr * ehdr;
```

```

Elf * elf;
Elf_Scn * scn;
Elf_Data * data;
int fd;
unsigned int cnt;
/* Open the input file */
if ((fd = open(argv[1], O_RDONLY)) == -1)
    exit(1);
/* Obtain the ELF descriptor */
(void) elf_version(EV_CURRENT);
if ((elf = elf_begin(fd, ELF_C_READ, NULL)) == NULL)
    failure();
/* Obtain the .shstrtab data buffer */
if ((ehdr = elf32_getehdr(elf)) == NULL) ||
    ((scn = elf_getscn(elf, ehdr->e_shstrndx)) == NULL) ||
    ((data = elf_getdata(scn, NULL)) == NULL))
    failure();
/* Traverse input filename, printing each section */
for (cnt = 1, scn = NULL; scn = elf_nextscn(elf, scn); cnt++) {
    if ((shdr = elf32_getshdr(scn)) == NULL)
        failure();
    (void) printf("[%d] %s\n", cnt,
        (char *)data->d_buf + shdr->sh_name);
}
/* end main */
static void
failure()
{
    (void) fprintf(stderr, "%s\n", elf_errmsg(elf_errno()));
    exit(1);
}

```

ATTRIBUTES

See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

[elf32_fsize\(3E\)](#), [elf32_getshdr\(3E\)](#), [elf32_xlatetof\(3E\)](#),
[elf_begin\(3E\)](#), [elf_cntl\(3E\)](#), [elf_errmsg\(3E\)](#), [elf_fill\(3E\)](#),
[elf_getarhdr\(3E\)](#), [elf_getarsym\(3E\)](#), [elf_getbase\(3E\)](#),
[elf_getdata\(3E\)](#), [elf_getident\(3E\)](#), [elf_getscn\(3E\)](#), [elf_hash\(3E\)](#),
[elf_kind\(3E\)](#), [elf_memory\(3E\)](#), [elf_rawfile\(3E\)](#), [elf_strpstr\(3E\)](#),
[elf_update\(3E\)](#), [elf_version\(3E\)](#), [ar\(4\)](#), [attributes\(5\)](#)

ANSI C Programmer's Guide

SPARC only

[a.out\(4\)](#)

NOTES

Information in the ELF headers is separated into common parts and processor-specific parts. A program can make a processor's information

available by including the appropriate header: `<sys/elf_NAME.h>` where *NAME* matches the processor name as used in the ELF file header.

```
Name Processor
M32 AT&T WE 32100
SPARC SPARC
386 Intel 80386, 80486, Pentium
```

Other processors will be added to the table as necessary.

To illustrate, a program could use the following code to “see” the processor-specific information for the SPARC based system.

```
#include <libelf.h>
#include <sys/elf_SPARC.h>
```

Without the `<sys/elf_SPARC.h>` definition, only the common ELF information would be visible.

A program could use the following code to “see” the processor-specific information for the Intel 80386:

```
#include <libelf.h>
#include <sys/elf_386.h>
```

Without the `<sys/elf_386.h>` definition, only the common ELF information would be visible.

Although reading the objects is rather straightforward, writing/updating them can corrupt the shared offsets among sections. Upon creation, relationships are established among the sections that must be maintained even if the object’s size is changed.

| | | | | | |
|--------------------|---|------------|---|------------|---|
| NAME | elf_begin, elf_end, elf_memory, elf_next, elf_rand – process ELF object files | | | | |
| SYNOPSIS | <pre> cc [flag ...] file ... -lelf [library ...] #include <libelf.h> Elf * elf_begin(int fildes, Elf_Cmd cmd, Elf * ref); int elf_end(Elf * elf); Elf * elf_memory(char * image, size_t sz); Elf_Cmd elf_next(Elf * elf); size_t elf_rand(Elf * elf, size_t offset); </pre> | | | | |
| DESCRIPTION | <p>elf_begin() , elf_end() , elf_memory() , elf_next() , and elf_rand() work together to process Executable and Linking Format (ELF) object files, either individually or as members of archives. After obtaining an ELF descriptor from elf_begin() or elf_memory() , the program may read an existing file, update an existing file, or create a new file. <i>fildes</i> is an open file descriptor that elf_begin() uses for reading or writing. <i>elf</i> is an ELF descriptor previously returned from elf_begin() . The initial file offset (see lseek(2)) is unconstrained, and the resulting file offset is undefined.</p> <p><i>cmd</i> may have the following values:</p> <table border="0"> <tr> <td style="padding-right: 20px;">ELF_C_NULL</td> <td>When a program sets <i>cmd</i> to this value, elf_begin() returns a null pointer, without opening a new descriptor. <i>ref</i> is ignored for this command. See the examples below for more information.</td> </tr> <tr> <td>ELF_C_READ</td> <td>When a program wishes to examine the contents of an existing file, it should set <i>cmd</i> to this value. Depending on the value of <i>ref</i> , this command examines archive members or entire files. Three cases can occur.</td> </tr> </table> <p>First, if <i>ref</i> is a null pointer, elf_begin() allocates a new ELF descriptor and prepares to process the entire file. If the file being read is an archive, elf_begin() also prepares the resulting descriptor to examine the initial archive member on</p> | ELF_C_NULL | When a program sets <i>cmd</i> to this value, elf_begin() returns a null pointer, without opening a new descriptor. <i>ref</i> is ignored for this command. See the examples below for more information. | ELF_C_READ | When a program wishes to examine the contents of an existing file, it should set <i>cmd</i> to this value. Depending on the value of <i>ref</i> , this command examines archive members or entire files. Three cases can occur. |
| ELF_C_NULL | When a program sets <i>cmd</i> to this value, elf_begin() returns a null pointer, without opening a new descriptor. <i>ref</i> is ignored for this command. See the examples below for more information. | | | | |
| ELF_C_READ | When a program wishes to examine the contents of an existing file, it should set <i>cmd</i> to this value. Depending on the value of <i>ref</i> , this command examines archive members or entire files. Three cases can occur. | | | | |

the next call to **elf_begin()** , as if the program had used **elf_next()** or **elf_rand()** to “move” to the initial member.

Second, if *ref* is a non-null descriptor associated with an archive file, **elf_begin()** lets a program obtain a separate ELF descriptor associated with an individual member. The program should have used **elf_next()** or **elf_rand()** to position *ref* appropriately (except for the initial member, which **elf_begin()** prepares; see the example below). In this case, *files* should be the same file descriptor used for the parent archive.

Finally, if *ref* is a non-null ELF descriptor that is not an archive, **elf_begin()** increments the number of activations for the descriptor and returns *ref* , without allocating a new descriptor and without changing the descriptor’s read/write permissions. To terminate the descriptor for *ref* , the program must call **elf_end()** once for each activation. See the examples below for more information.

ELF_C_RDWR This command duplicates the actions of **ELF_C_READ** and additionally allows the program to update the file image (see **elf_update(3E)**). That is, using **ELF_C_READ** gives a read-only view of the file, while **ELF_C_RDWR** lets the program read *and* write the file. **ELF_C_RDWR** is not valid for archive members. If *ref* is non-null, it must have been created with the **ELF_C_RDWR** command.

ELF_C_WRITE If the program wishes to ignore previous file contents, presumably to create a new file, it should set *cmd* to this value. *ref* is ignored for this command.

elf_begin() “works” on all files (including files with zero bytes), providing it can allocate memory for its internal structures and read any necessary information from the file. Programs reading object files thus may call **elf_kind(3E)** or **elf32_getehdr(3E)** to determine the file type (only object files have an ELF header). If the file is an archive with no more members to process, or an error occurs, **elf_begin()** returns a null pointer. Otherwise, the return value is a non-null ELF descriptor.

Before the first call to **elf_begin()** , a program must call **elf_version()** to coordinate versions.

elf_end() is used to terminate an ELF descriptor, *elf* , and to deallocate data associated with the descriptor. Until the program terminates a descriptor, the data remain allocated. A null pointer is allowed as an argument, to simplify error handling. If the program wishes to write data associated with the ELF descriptor to the file, it must use **elf_update()** before calling **elf_end()** .

Calling **elf_end()** removes one activation and returns the remaining activation count. The library does not terminate the descriptor until the activation count reaches 0. Consequently, a 0 return value indicates the ELF descriptor is no longer valid.

elf_memory() returns a pointer to an ELF descriptor, the ELF image has read operations enabled (`ELF_C_READ`). *image* is a pointer to an image of the Elf file mapped into memory, *sz* is the size of the ELF image. An ELF image that is mapped in with **elf_memory()** may be read and modified, but the ELF image size may not be changed.

elf_next() provides sequential access to the next archive member. That is, having an ELF descriptor, *elf*, associated with an archive member, **elf_next()** prepares the containing archive to access the following member when the program calls **elf_begin()**. After successfully positioning an archive for the next member, **elf_next()** returns the value `ELF_C_READ`. Otherwise, the open file was not an archive, *elf* was `NULL`, or an error occurred, and the return value is `ELF_C_NULL`. In either case, the return value may be passed as an argument to **elf_begin()**, specifying the appropriate action.

elf_rand() provides random archive processing, preparing *elf* to access an arbitrary archive member. *elf* must be a descriptor for the archive itself, not a member within the archive. *offset* gives the byte offset from the beginning of the archive to the archive header of the desired member. See **elf_getarsym(3E)** for more information about archive member offsets. When **elf_rand()** works, it returns *offset*. Otherwise, it returns 0, because an error occurred, *elf* was `NULL`, or the file was not an archive (no archive member can have a zero offset). A program may mix random and sequential archive processing.

System Services

When processing a file, the library decides when to read or write the file, depending on the program's requests. Normally, the library assumes the file descriptor remains usable for the life of the ELF descriptor. If, however, a program must process many files simultaneously and the underlying operating system limits the number of open files, the program can use **elf_cntl()** to let it reuse file descriptors. After calling **elf_cntl()** with appropriate arguments, the program may close the file descriptor without interfering with the library.

All data associated with an ELF descriptor remain allocated until **elf_end()** terminates the descriptor's last activation. After the descriptors have been terminated, the storage is released; attempting to reference such data gives undefined behavior. Consequently, a program that deals with multiple input (or output) files must keep the ELF descriptors active until it finishes with them.

EXAMPLES

EXAMPLE 1 A sample program of calling the **elf_begin()** function.

A prototype for reading a file appears on the next page. If the file is a simple object file, the program executes the loop one time, receiving a null descriptor

in the second iteration. In this case, both `elf` and `arf` will have the same value, the activation count will be 2, and the program calls `elf_end()` twice to terminate the descriptor. If the file is an archive, the loop processes each archive member in turn, ignoring those that are not object files.

```

if (elf_version(EV_CURRENT) == EV_NONE)
{
\011/* library out of date */
\011/* recover from error */
}
cmd = ELF_C_READ;
arf = elf_begin(fildes, cmd, (Elf *)0);
while ((elf = elf_begin(fildes, cmd, arf)) != 0)
{
\011if ((ehdr = elf32_getehdr(elf)) != 0)
\011{
\011\011/* process the file ... */
\011}
\011cmd = elf_next(elf);
\011elf_end(elf);
}
elf_end(arf);

```

Alternatively, the next example illustrates random archive processing. After identifying the file as an archive, the program repeatedly processes archive members of interest. For clarity, this example omits error checking and ignores simple object files. Additionally, this fragment preserves the ELF descriptors for all archive members, because it does not call `elf_end()` to terminate them.

```

elf_version(EV_CURRENT);
arf = elf_begin(fildes, ELF_C_READ, (Elf *)0);
if (elf_kind(arf) != ELF_K_AR)
{
\011/* not an archive */
}
/* initial processing */
/* set offset = ... for desired member header */
while (elf_rand(arf, offset) == offset)
{
\011if ((elf = elf_begin(fildes, ELF_C_READ, arf)) == 0)
\011\011break;
\011if ((ehdr = elf32_getehdr(elf)) != 0)
\011{
\011\011/* process archive member ... */
\011}
\011/* set offset = ... for desired member header */
}

```

An archive starts with a “magic string” that has SARMAG bytes; the initial archive member follows immediately. An application could thus provide the

following function to rewind an archive (the function returns `-1` for errors and `0` otherwise).

```
#include <ar.h>
#include <libelf.h>
int
rewindelf(Elf *elf)
{
    if (elf_rand(elf, (size_t)SARMAG) == SARMAG)
        return 0;
    return -1;
}
```

The following outline shows how one might create a new ELF file. This example is simplified to show the overall flow.

```
elf_version(EV_CURRENT);
fildes = open("path/name", O_RDWR|O_TRUNC|O_CREAT, 0666);
if ((elf = elf_begin(fildes, ELF_C_WRITE, (Elf *)0)) == 0)
    \011return;
ehdr = elf32_newehdr(elf);
phdr = elf32_newphdr(elf, count);
scn = elf_newscn(elf);
shdr = elf32_getshdr(scn);
data = elf_newdata(scn);
elf_update(elf, ELF_C_WRITE);
elf_end(elf);
```

Finally, the following outline shows how one might update an existing ELF file. Again, this example is simplified to show the overall flow.

```
elf_version(EV_CURRENT);
fildes = open("path/name", O_RDWR);
elf = elf_begin(fildes, ELF_C_RDWR, (Elf *)0);
/* add new or delete old information */
...
/* ensure that the memory image of the file is complete */
elf_update(elf, ELF_C_NULL);
elf_update(elf, ELF_C_WRITE); /* update file */
elf_end(elf);
```

Notice that both file creation examples open the file with write *and* read permissions. On systems that support `mmap(2)`, the library uses it to enhance performance, and `mmap(2)` requires a readable file descriptor. Although the library can use a write-only file descriptor, the application will not obtain the performance advantages of `mmap(2)`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`creat(2)`, `lseek(2)`, `mmap(2)`, `open(2)`, `elf(3E)`, `elf32_getehdr(3E)`, `elf_cntl(3E)`, `elf_getarhdr(3E)`, `elf_getarsym(3E)`, `elf_getbase(3E)`, `elf_getdata(3E)`, `elf_getscn(3E)`, `elf_kind(3E)`, `elf_rawfile(3E)`, `elf_update(3E)`, `elf_version(3E)`, `ar(4)`, `attributes(5)`

NAME elf_cntl – control an elf file descriptor

SYNOPSIS

```
cc [ flag ... ] file ... -lelf [ library ... ]
#include <libelf.h>

int elf_cntl(Elf *elf, Elf_Cmd cmd);
```

DESCRIPTION `elf_cntl()` instructs the library to modify its behavior with respect to an ELF descriptor, *elf*. As `elf_begin(3E)` describes, an ELF descriptor can have multiple activations, and multiple ELF descriptors may share a single file descriptor. Generally, `elf_cntl()` commands apply to all activations of *elf*. Moreover, if the ELF descriptor is associated with an archive file, descriptors for members within the archive will also be affected as described below. Unless stated otherwise, operations on archive members do not affect the descriptor for the containing archive.

The *cmd* argument tells what actions to take and may have the following values:

`ELF_C_FDDONE` This value tells the library not to use the file descriptor associated with *elf*. A program should use this command when it has requested all the information it cares to use and wishes to avoid the overhead of reading the rest of the file. The memory for all completed operations remains valid, but later file operations, such as the initial `elf_getdata()` for a section, will fail if the data are not in memory already.

`ELF_C_FDREAD` This command is similar to `ELF_C_FDDONE`, except it forces the library to read the rest of the file. A program should use this command when it must close the file descriptor but has not yet read everything it needs from the file. After `elf_cntl()` completes the `ELF_C_FDREAD` command, future operations, such as `elf_getdata()`, will use the memory version of the file without needing to use the file descriptor.

If `elf_cntl()` succeeds, it returns 0. Otherwise *elf* was NULL or an error occurred, and the function returns -1.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO `elf(3E)`, `elf_begin(3E)`, `elf_getdata(3E)`, `elf_rawfile(3E)`, `attributes(5)`

NOTES

If the program wishes to use the “raw” operations (see **elf_rawdata()**, which **elf_getdata(3E)** describes, and **elf_rawfile(3E)**) after disabling the file descriptor with `ELF_C_FDDONE` or `ELF_C_FDREAD`, it must execute the raw operations explicitly beforehand. Otherwise, the raw file operations will fail. Calling **elf_rawfile()** makes the entire image available, thus supporting subsequent **elf_rawdata()** calls.

| | |
|--------------------|---|
| NAME | elf_errmsg, elf_errno – error handling |
| SYNOPSIS | <pre>cc [flag ...] file ... -lelf [library ...] #include <libelf.h> const char * elf_errmsg(int err); int elf_errno(void);</pre> |
| DESCRIPTION | <p>If an ELF library function fails, a program may call elf_errno() to retrieve the library's internal error number. As a side effect, this function resets the internal error number to 0 , which indicates no error.</p> <p>elf_errmsg() takes an error number, <i>err</i> , and returns a null-terminated error message (with no trailing new-line) that describes the problem. A zero <i>err</i> retrieves a message for the most recent error. If no error has occurred, the return value is a null pointer (not a pointer to the null string). Using <i>err</i> of -1 also retrieves the most recent error, except it guarantees a non-null return value, even when no error has occurred. If no message is available for the given number, elf_errmsg() returns a pointer to an appropriate message. This function does not have the side effect of clearing the internal error number.</p> |
| EXAMPLES | <p>EXAMPLE 1 A sample program of calling the elf_errmsg() function.</p> <p>The following fragment clears the internal error number and checks it later for errors. Unless an error occurs after the first call to elf_errno() , the next call will return 0 .</p> <pre>(void)elf_errno(); /* processing ... */ while (more_to_do) { \011if ((err = elf_errno()) != 0) \011{ \011\011/* print msg */ \011\011msg = elf_errmsg(err); \011} }</pre> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

elf(3E) , **attributes(5)**

NAME elf_fill – set fill byte

SYNOPSIS

```
cc [ flag ... ] file ... -lelf [ library ... ]
#include <libelf.h>

void elf_fill(int fill);
```

DESCRIPTION Alignment constraints for ELF files sometimes require the presence of “holes.” For example, if the data for one section are required to begin on an eight-byte boundary, but the preceding section is too “short,” the library must fill the intervening bytes. These bytes are set to the *fill* character. The library uses zero bytes unless the application supplies a value. See [elf_getdata\(3E\)](#) for more information about these holes.

ATTRIBUTES See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO [elf\(3E\)](#), [elf_flagdata\(3E\)](#), [elf_getdata\(3E\)](#), [elf_update\(3E\)](#), [attributes\(5\)](#)

NOTES An application can assume control of the object file organization by setting the `ELF_F_LAYOUT` bit (see [elf_flagdata\(3E\)](#)). When this is done, the library does *not* fill holes.

| | |
|--------------------|--|
| NAME | elf_flagdata, elf_flagehdr, elf_flagelf, elf_flagphdr, elf_flagscn, elf_flagshdr – manipulate flags |
| SYNOPSIS | <pre>cc [flag ...] file ... -lelf [library ...] #include <libelf.h> unsigned elf_flagdata(Elf_Data * data, Elf_Cmd cmd, unsigned flags); unsigned elf_flagehdr(Elf * elf, Elf_Cmd cmd, unsigned flags); unsigned elf_flagelf(Elf * elf, Elf_Cmd cmd, unsigned flags); unsigned elf_flagphdr(Elf * elf, Elf_Cmd cmd, unsigned flags); unsigned elf_flagscn(Elf_Scn * scn, Elf_Cmd cmd, unsigned flags); unsigned elf_flagshdr(Elf_Scn * scn, Elf_Cmd cmd, unsigned flags);</pre> |
| DESCRIPTION | <p>These functions manipulate the flags associated with various structures of an ELF file. Given an ELF descriptor (<i>elf</i>), a data descriptor (<i>data</i>), or a section descriptor (<i>scn</i>), the functions may set or clear the associated status bits, returning the updated bits. A null descriptor is allowed, to simplify error handling; all functions return 0 for this degenerate case.</p> <p><i>cmd</i> may have the following values:</p> <p>ELF_C_CLR The functions clear the bits that are asserted in <i>flags</i> . Only the non-zero bits in <i>flags</i> are cleared; zero bits do not change the status of the descriptor.</p> <p>ELF_C_SET The functions set the bits that are asserted in <i>flags</i> . Only the non-zero bits in <i>flags</i> are set; zero bits do not change the status of the descriptor.</p> <p>Descriptions of the defined <i>flags</i> bits appear below:</p> <p>ELF_F_DIRTY When the program intends to write an ELF file, this flag asserts the associated information needs to be written to the file. Thus, for example, a program that wished to update the ELF header of an existing file would call elf_flagehdr() with this bit set in <i>flags</i> and <i>cmd</i> equal to ELF_C_SET . A later call to elf_update() would write the marked header to the file.</p> |

ELF_F_LAYOUT Normally, the library decides how to arrange an output file. That is, it automatically decides where to place sections, how to align them in the file, etc. If this bit is set for an ELF descriptor, the program assumes responsibility for determining all file positions. This bit is meaningful only for **elf_flagelf()** and applies to the entire file associated with the descriptor.

When a flag bit is set for an item, it affects all the subitems as well. Thus, for example, if the program sets the **ELF_F_DIRTY** bit with **elf_flagelf()**, the entire logical file is “dirty.”

EXAMPLES

EXAMPLE 1 A sample display of calling the **elf_flagdata()** function.

The following fragment shows how one might mark the ELF header to be written to the output file:

```
/* dirty ehdr ... */
ehdr = elf32_getehdr(elf);
elf_flagehdr(elf, ELF_C_SET, ELF_F_DIRTY);
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

elf(3E), **elf32_getehdr(3E)**, **elf_getdata(3E)**, **elf_update(3E)**, **attributes(5)**

| | |
|--------------------|--|
| NAME | elf_getarhdr – retrieve archive member header |
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lelf [<i>library...</i>] #include <libelf.h> Elf_Arhdr *elf_getarhdr(Elf *elf);</pre> |
| DESCRIPTION | <p>elf_getarhdr() returns a pointer to an archive member header, if one is available for the ELF descriptor <i>elf</i>. Otherwise, no archive member header exists, an error occurred, or <i>elf</i> was null; elf_getarhdr() then returns a null value. The header includes the following members.</p> <pre>char *ar_name; time_t ar_date; uid_t ar_uid; gid_t ar_gid; mode_t ar_mode; off_t ar_size; char *ar_rawname;</pre> <p>An archive member name, available through <i>ar_name</i>, is a null-terminated string, with the <i>ar</i> format control characters removed. The <i>ar_rawname</i> member holds a null-terminated string that represents the original name bytes in the file, including the terminating slash and trailing blanks as specified in the archive format.</p> <p>In addition to “regular” archive members, the archive format defines some special members. All special member names begin with a slash (/), distinguishing them from regular members (whose names may not contain a slash). These special members have the names (<i>ar_name</i>) defined below.</p> <pre>/</pre> <p>This is the archive symbol table. If present, it will be the first archive member. A program may access the archive symbol table through elf_getarsym(). The information in the symbol table is useful for random archive processing (see elf_rand() on elf_begin(3E)).</p> <pre>//</pre> <p>This member, if present, holds a string table for long archive member names. An archive member’s header contains a 16-byte area for the name, which may be exceeded in some file systems. The library automatically retrieves long member names from the string table, setting <i>ar_name</i> to the appropriate value.</p> <p>Under some error conditions, a member’s name might not be available. Although this causes the library to set <i>ar_name</i> to a null pointer, the <i>ar_rawname</i> member will be set as usual.</p> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO `elf(3E)`, `elf_begin(3E)`, `elf_getarsym(3E)`, `ar(4)`, `attributes(5)`

| NAME | elf_getarsym – retrieve archive symbol table | | | | |
|--------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lelf [<i>library ...</i>] #include <libelf.h> Elf_Arsym *elf_getarsym(Elf *elf, size_t *ptr);</pre> | | | | |
| DESCRIPTION | <p>elf_getarsym() returns a pointer to the archive symbol table, if one is available for the ELF descriptor <i>elf</i>. Otherwise, the archive doesn't have a symbol table, an error occurred, or <i>elf</i> was null; elf_getarsym() then returns a null value. The symbol table is an array of structures that include the following members.</p> <pre>char *as_name; size_t as_off; unsigned long as_hash;</pre> <p>These members have the following semantics:</p> <p>as_name A pointer to a null-terminated symbol name resides here.</p> <p>as_off This value is a byte offset from the beginning of the archive to the member's header. The archive member residing at the given offset defines the associated symbol. Values in as_off may be passed as arguments to elf_rand(). See elf_begin(3E) to access the desired archive member.</p> <p>as_hash This is a hash value for the name, as computed by elf_hash().</p> <p>If <i>ptr</i> is non-null, the library stores the number of table entries in the location to which <i>ptr</i> points. This value is set to 0 when the return value is NULL. The table's last entry, which is included in the count, has a null as_name, a zero value for as_off, and ~0UL for as_hash.</p> <p>The hash value returned is guaranteed not to be the bit pattern of all ones (~0UL).</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | elf(3E) , elf_begin(3E) , elf_getarhdr(3E) , elf_hash(3E) , ar(4) , attributes(5) | | | | |

NAME elf_getbase – get the base offset for an object file

SYNOPSIS cc [*flag* ...] *file* ... -lelf [*library* ...]
 #include <libelf.h>
 off_t elf_getbase(Elf *elf);

DESCRIPTION **elf_getbase()** returns the file offset of the first byte of the file or archive member associated with *elf*, if it is known or obtainable, and -1 otherwise. A null *elf* is allowed, to simplify error handling; the return value in this case is -1 . The base offset of an archive member is the beginning of the member's information, *not* the beginning of the archive member header.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **elf(3E)**, **elf_begin(3E)**, **ar(4)**, **attributes(5)**

| | |
|--------------------|--|
| NAME | elf_getdata, elf_newdata, elf_rawdata – get section data |
| SYNOPSIS | <pre>cc [flag ...] file ... -lelf [library ...] #include <libelf.h> Elf_Data * elf_getdata(Elf_Scn * scn, Elf_Data * data); Elf_Data * elf_newdata(Elf_Scn * scn); Elf_Data * elf_rawdata(Elf_Scn * scn, Elf_Data * data);</pre> |
| DESCRIPTION | <p>These functions access and manipulate the data associated with a section descriptor, <i>scn</i>. When reading an existing file, a section will have a single data buffer associated with it. A program may build a new section in pieces, however, composing the new data from multiple data buffers. For this reason, the data for a section should be viewed as a list of buffers, each of which is available through a data descriptor.</p> <p>elf_getdata() lets a program step through a section's data list. If the incoming data descriptor, <i>data</i>, is null, the function returns the first buffer associated with the section. Otherwise, <i>data</i> should be a data descriptor associated with <i>scn</i>, and the function gives the program access to the next data element for the section. If <i>scn</i> is null or an error occurs, elf_getdata() returns a null pointer.</p> <p>elf_getdata() translates the data from file representations into memory representations (see elf32_xlatetof(3E)) and presents objects with memory data types to the program, based on the file's <i>class</i> (see elf(3E)). The working library version (see elf_version(3E)) specifies what version of the memory structures the program wishes elf_getdata() to present.</p> <p>elf_newdata() creates a new data descriptor for a section, appending it to any data elements already associated with the section. As described below, the new data descriptor appears empty, indicating the element holds no data. For convenience, the descriptor's type (<i>d_type</i> below) is set to <code>ELF_T_BYTE</code>, and the version (<i>d_version</i> below) is set to the working version. The program is responsible for setting (or changing) the descriptor members as needed. This function implicitly sets the <code>ELF_F_DIRTY</code> bit for the section's data (see elf_flagdata(3E)). If <i>scn</i> is null or an error occurs, elf_newdata() returns a null pointer.</p> |

elf_rawdata() differs from **elf_getdata()** by returning only uninterpreted bytes, regardless of the section type. This function typically should be used only to retrieve a section image from a file being read, and then only when a program must avoid the automatic data translation described below. Moreover, a program may not close or disable (see **elf_cntl(3E)**) the file descriptor associated with *elf* before the initial raw operation, because **elf_rawdata()** might read the data from the file to ensure it doesn't interfere with **elf_getdata()**. See **elf_rawfile(3E)** for a related facility that applies to the entire file. When **elf_getdata()** provides the right translation, its use is recommended over **elf_rawdata()**. If *scn* is null or an error occurs, **elf_rawdata()** returns a null pointer.

The `Elf_Data` structure includes the following members:

```
\011void\011*d_buf;
\011Elf_Type\011d_type;
\011size_t\011d_size;
\011off_t\011d_off;
\011size_t\011d_align;
\011unsigned\011d_version;
```

These members are available for direct manipulation by the program. Descriptions appear below.

| | |
|----------------------|---|
| <code>d_buf</code> | A pointer to the data buffer resides here. A data element with no data has a null pointer. |
| <code>d_type</code> | This member's value specifies the type of the data to which <code>d_buf</code> points. A section's type determines how to interpret the section contents, as summarized below. |
| <code>d_size</code> | This member holds the total size, in bytes, of the memory occupied by the data. This may differ from the size as represented in the file. The size will be zero if no data exist. (See the discussion of <code>SHT_NOBITS</code> below for more information.) |
| <code>d_off</code> | This member gives the offset, within the section, at which the buffer resides. This offset is relative to the file's section, not the memory object's. |
| <code>d_align</code> | This member holds the buffer's required alignment, from the beginning of the section. That is, <code>d_off</code> will be a multiple of this member's value. For example, if this member's value is 4, the beginning of the buffer will be four-byte aligned within |

| | |
|--------------------------|--|
| Data Alignment | <p>the section. Moreover, the entire section will be aligned to the maximum of its constituents, thus ensuring appropriate alignment for a buffer within the section and within the file.</p> <p><code>d_version</code> This member holds the version number of the objects in the buffer. When the library originally read the data from the object file, it used the working version to control the translation to memory objects.</p> <p>As mentioned above, data buffers within a section have explicit alignment constraints. Consequently, adjacent buffers sometimes will not abut, causing “holes” within a section. Programs that create output files have two ways of dealing with these holes.</p> <p>First, the program can use <code>elf_fill()</code> to tell the library how to set the intervening bytes. When the library must generate gaps in the file, it uses the fill byte to initialize the data there. The library’s initial fill value is 0 , and <code>elf_fill()</code> lets the application change that.</p> <p>Second, the application can generate its own data buffers to occupy the gaps, filling the gaps with values appropriate for the section being created. A program might even use different fill values for different sections. For example, it could set text sections’ bytes to no-operation instructions, while filling data section holes with zero. Using this technique, the library finds no holes to fill, because the application eliminated them.</p> |
| Section and Memory Types | <p><code>elf_getdata()</code> interprets sections’ data according to the section type, as noted in the section header available through <code>elf32_getshdr()</code> . The following table shows the section types and how the library represents them with memory data types for the 32-bit file class. Other classes would have similar tables. By implication, the memory data types control translation by <code>elf32_xlatetof(3E)</code></p> <pre> Section Type\011Elf_Type\01132-Bit Type SHT_DYNAMIC\011ELF_T_DYN\011Elf32_Dyn SHT_DYNSYM\011ELF_T_SYM\011Elf32_Sym SHT_HASH\011ELF_T_WORD\011Elf32_Word SHT_NOBITS\011ELF_T_BYTE\011unsigned char SHT_NOTE\011ELF_T_BYTE\011unsigned char SHT_NULL\011 none \011 none SHT_PROGBITS\011ELF_T_BYTE\011unsigned char SHT_REL\011ELF_T_REL\011Elf32_Rel SHT_RELA\011ELF_T_RELA\011Elf32_Rela SHT_STRTAB\011ELF_T_BYTE\011unsigned char SHT_SYMTAB\011ELF_T_SYM\011Elf32_Sym </pre> |

```
SHT_SUNW_verdef\011ELF_T_VDEF\011Elf32_Verdef
SHT_SUNW_verneed\011ELF_T_VNEED\011Elf32_Verneed
SHT_SUNW_versym\011ELF_T_HALF\011Elf32_Versym
```

other
\011ELF_T_BYTE\011unsigned char

elf_rawdata() creates a buffer with type `ELF_T_BYTE` .

As mentioned above, the program's working version controls what structures the library creates for the application. The library similarly interprets section types according to the versions. If a section type belongs to a version newer than the application's working version, the library does not translate the section data. Because the application cannot know the data format in this case, the library presents an untranslated buffer of type `ELF_T_BYTE` , just as it would for an unrecognized section type.

A section with a special type, `SHT_NOBITS` , occupies no space in an object file, even when the section header indicates a non-zero size. **elf_getdata()** and **elf_rawdata()** work on such a section, setting the *data* structure to have a null buffer pointer and the type indicated above. Although no data are present, the `d_size` value is set to the size from the section header. When a program is creating a new section of type `SHT_NOBITS` , it should use **elf_newdata()** to add data buffers to the section. These empty data buffers should have the `d_size` members set to the desired size and the `d_buf` members set to `NULL` .

EXAMPLES

EXAMPLE 1 A sample program of calling **elf_getdata()** .

The following fragment obtains the string table that holds section names (ignoring error checking). See **elf_strptr(3E)** for a variation of string table handling.

```
ehdr = elf32_getehdr(elf);
scn = elf_getscn(elf, (size_t)ehdr⇒e_shstrndx);
shdr = elf32_getshdr(scn);
if (shdr⇒sh_type != SHT_STRTAB)
{
\011/* not a string table */
}
data = 0;
if ((data = elf_getdata(scn, data)) == 0 || data⇒d_size == 0)
{
\011/* error or no data */
}
```

The `e_shstrndx` member in an ELF header holds the section table index of the string table. The program gets a section descriptor for that section, verifies it is a string table, and then retrieves the data. When this fragment finishes, `data⇒d_buf` points at the first byte of the string table, and `data⇒d_size` holds the string table's size in bytes.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`elf(3E)`, `elf32_getehdr(3E)`, `elf32_getshdr(3E)`,
`elf32_xlatetof(3E)`, `elf_cntl(3E)`, `elf_fill(3E)`,
`elf_flagdata(3E)`, `elf_getscn(3E)`, `elf_rawfile(3E)`,
`elf_strptr(3E)`, `elf_version(3E)`, `attributes(5)`

NAME elf_getident – retrieve file identification data

SYNOPSIS cc [*flag ...*] *file ...* -lelf [*library ...*]
#include <libelf.h>

```
char *elf_getident(Elf *elf, size_t *ptr);
```

DESCRIPTION As [elf\(3E\)](#) explains, ELF provides a framework for various classes of files, where basic objects may have 32 bits, 64 bits, etc. To accommodate these differences, without forcing the larger sizes on smaller machines, the initial bytes in an ELF file hold identification information common to all file classes. Every ELF header's `e_ident` has `EI_NIDENT` bytes with the following interpretation:

| e_ident Index | Value | Purpose |
|---------------|--------------|---------------------|
| EI_MAG0 | ELFMAG0 | File identification |
| EI_MAG1 | ELFMAG1 | |
| EI_MAG2 | ELFMAG2 | |
| EI_MAG3 | ELFMAG3 | |
| EI_CLASS | ELFCLASSNONE | File class |
| | ELFCLASS32 | |
| | ELFCLASS64 | |
| EI_DATA | ELFDATANONE | Data encoding |
| | ELFDATA2LSB | |
| | ELFDATA2MSB | |
| EI_VERSION | EV_CURRENT | File version |
| 7-15 | 0 | Unused, set to zero |

Other kinds of files (see [elf_kind\(3E\)](#)) also may have identification data, though they would not conform to `e_ident`.

elf_getident() returns a pointer to the file's "initial bytes." If the library recognizes the file, a conversion from the file image to the memory image may occur. In any case, the identification bytes are guaranteed not to have been

modified, though the size of the unmodified area depends on the file type. If *ptr* is non-null, the library stores the number of identification bytes in the location to which *ptr* points. If no data are present, *elf* is null, or an error occurs, the return value is a null pointer, with 0 stored through *ptr*, if *ptr* is non-null.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

elf(3E), **elf32_getehdr(3E)**, **elf_begin(3E)**, **elf_kind(3E)**, **elf_rawfile(3E)**, **attributes(5)**

| | |
|--------------------|---|
| NAME | elf_getscn, elf_ndxscn, elf_newscn, elf_nextscn – get section information |
| SYNOPSIS | <pre>cc [flag ...] file ... -lelf [library ...] #include <libelf.h> Elf_Scn *elf_getscn(Elf *elf, size_t index); size_t elf_ndxscn(Elf_Scn *scn); Elf_Scn *elf_newscn(Elf *elf); Elf_Scn *elf_nextscn(Elf *elf, Elf_Scn *scn);</pre> |
| DESCRIPTION | <p>These functions provide indexed and sequential access to the sections associated with the ELF descriptor <i>elf</i>. If the program is building a new file, it is responsible for creating the file's ELF header before creating sections; see elf32_getehdr(3E).</p> <p>elf_getscn() returns a section descriptor, given an <i>index</i> into the file's section header table. Note that the first "real" section has an index of 1. Although a program can get a section descriptor for the section whose <i>index</i> is 0 (SHN_UNDEF, the undefined section), the section has no data and the section header is "empty" (though present). If the specified section does not exist, an error occurs, or <i>elf</i> is null, elf_getscn() returns a null pointer.</p> <p>elf_newscn() creates a new section and appends it to the list for <i>elf</i>. Because the SHN_UNDEF section is required and not "interesting" to applications, the library creates it automatically. Thus the first call to elf_newscn() for an ELF descriptor with no existing sections returns a descriptor for section 1. If an error occurs or <i>elf</i> is null, elf_newscn() returns a null pointer.</p> <p>After creating a new section descriptor, the program can use elf32_getshdr() to retrieve the newly created, "clean" section header. The new section descriptor will have no associated data (see elf_getdata(3E)). When creating a new section in this way, the library updates the <i>e_shnum</i> member of the ELF header and sets the ELF_F_DIRTY bit for the section (see elf_flagdata(3E)). If the program is building a new file, it is responsible for creating the file's ELF header (see elf32_getehdr(3E)) before creating new sections.</p> <p>elf_nextscn() takes an existing section descriptor, <i>scn</i>, and returns a section descriptor for the next higher section. One may use a null <i>scn</i> to obtain a</p> |

section descriptor for the section whose index is 1 (skipping the section whose index is `SHN_UNDEF`). If no further sections are present or an error occurs, `elf_nextscn()` returns a null pointer.

`elf_ndxscn()` takes an existing section descriptor, `scn`, and returns its section table index. If `scn` is null or an error occurs, `elf_ndxscn()` returns `SHN_UNDEF`.

EXAMPLES

EXAMPLE 1 A sample of calling `elf_getscn()` function.

An example of sequential access appears below. Each pass through the loop processes the next section in the file; the loop terminates when all sections have been processed.

```
scn = 0;
while ((scn = elf_nextscn(elf, scn)) != 0)
{
    \011/* process section */
}
```

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`elf(3E)`, `elf32_getehdr(3E)`, `elf32_getshdr(3E)`, `elf_begin(3E)`, `elf_flagdata(3E)`, `elf_getdata(3E)`, `attributes(5)`

NAME elf_hash – compute hash value

SYNOPSIS cc [*flag* ...] *file* ... -l`elf` [*library* ...]
#include <libelf.h>

unsigned long `elf_hash`(const char **name*);

DESCRIPTION `elf_hash()` computes a hash value, given a null terminated string, *name*. The returned hash value, *h*, can be used as a bucket index, typically after computing $h \bmod x$ to ensure appropriate bounds.

Hash tables may be built on one machine and used on another because `elf_hash()` uses unsigned arithmetic to avoid possible differences in various machines' signed arithmetic. Although *name* is shown as `char*` above, `elf_hash()` treats it as `unsigned char*` to avoid sign extension differences. Using `char*` eliminates type conflicts with expressions such as `elf_hash(name)`.

ELF files' symbol hash tables are computed using this function (see `elf_getdata(3E)` and `elf32_xlatetof(3E)`). The hash value returned is guaranteed not to be the bit pattern of all ones (`~0UL`).

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO `elf(3E)`, `elf32_xlatetof(3E)`, `elf_getdata(3E)`, `attributes(5)`

| NAME | elf_kind – determine file type | | | | |
|--------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lelf [library ...] #include <libelf.h> Elf_Kind elf_kind(Elf *elf);</pre> | | | | |
| DESCRIPTION | <p>This function returns a value identifying the kind of file associated with an ELF descriptor (<i>elf</i>). Defined values are below:</p> <p>ELF_K_AR The file is an archive [see ar(4)]. An ELF descriptor may also be associated with an archive <i>member</i>, not the archive itself, and then elf_kind() identifies the member's type.</p> <p>ELF_K_COFF The file is a COFF object file. elf_begin(3E) describes the library's handling for COFF files.</p> <p>ELF_K_ELF The file is an ELF file. The program may use elf_getident() to determine the class. Other functions, such as elf32_getehdr(), are available to retrieve other file information.</p> <p>ELF_K_NONE This indicates a kind of file unknown to the library. Other values are reserved, to be assigned as needed to new kinds of files. <i>elf</i> should be a value previously returned by elf_begin(). A null pointer is allowed, to simplify error handling, and causes elf_kind() to return ELF_K_NONE.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | elf(3E) , elf32_getehdr(3E) , elf_begin(3E) , elf_getident(3E) , ar(4) , attributes(5) | | | | |

NAME elf_rawfile – retrieve uninterpreted file contents

SYNOPSIS

```
cc [ flag... ] file ... -lelf [ library ... ]
#include <libelf.h>

char *elf_rawfile(Elf *elf, size_t *ptr);
```

DESCRIPTION

elf_rawfile() returns a pointer to an uninterpreted byte image of the file. This function should be used only to retrieve a file being read. For example, a program might use **elf_rawfile()** to retrieve the bytes for an archive member.

A program may not close or disable (see **elf_cntl(3E)**) the file descriptor associated with *elf* before the initial call to **elf_rawfile()**, because **elf_rawfile()** might have to read the data from the file if it does not already have the original bytes in memory. Generally, this function is more efficient for unknown file types than for object files. The library implicitly translates object files in memory, while it leaves unknown files unmodified. Thus, asking for the uninterpreted image of an object file may create a duplicate copy in memory.

elf_rawdata() is a related function, providing access to sections within a file. See **elf_getdata(3E)**.

If *ptr* is non-null, the library also stores the file's size, in bytes, in the location to which *ptr* points. If no data are present, *elf* is null, or an error occurs, the return value is a null pointer, with 0 stored through *ptr*, if *ptr* is non-null.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **elf(3E)**, **elf32_getehdr(3E)**, **elf_begin(3E)**, **elf_cntl(3E)**, **elf_getdata(3E)**, **elf_getident(3E)**, **elf_kind(3E)**, **attributes(5)**

NOTES

A program that uses **elf_rawfile()** and that also interprets the same file as an object file potentially has two copies of the bytes in memory. If such a program requests the raw image first, before it asks for translated information (through such functions as **elf32_getehdr()**, **elf_getdata()**, and so on), the library “freezes” its original memory copy for the raw image. It then uses this frozen copy as the source for creating translated objects, without reading the file again. Consequently, the application should view the raw file image returned by **elf_rawfile()** as a read-only buffer, unless it wants to alter its own view of data subsequently translated. In any case, the application may alter the translated objects without changing bytes visible in the raw image.

Multiple calls to **elf_rawfile()** with the same ELF descriptor return the same value; the library does not create duplicate copies of the file.

| NAME | elf_strptr – make a string pointer | | | | |
|--------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lelf [library ...] #include <libelf.h> char *elf_strptr(Elf *elf, size_t section, size_t offset);</pre> | | | | |
| DESCRIPTION | <p>This function converts a string section <i>offset</i> to a string pointer. <i>elf</i> identifies the file in which the string section resides, and <i>section</i> identifies the section table index for the strings. elf_strptr() normally returns a pointer to a string, but it returns a null pointer when <i>elf</i> is null, <i>section</i> is invalid or is not a section of type SHT_STRTAB, the section data cannot be obtained, <i>offset</i> is invalid, or an error occurs.</p> | | | | |
| EXAMPLES | <p>EXAMPLE 1 A sample program of calling elf_strptr() function.</p> <p>A prototype for retrieving section names appears below. The file header specifies the section name string table in the <code>e_shstrndx</code> member. The following code loops through the sections, printing their names.</p> <pre>/* handle the error */ if ((ehdr = elf32_getehdr(elf)) == 0) { return; } ndx = ehdr->e_shstrndx; scn = 0; while ((scn = elf_nextscn(elf, scn)) != 0) { char *name = 0; if ((shdr = elf32_getshdr(scn)) != 0) name = elf_strptr(elf, ndx, (size_t)shdr->sh_name); printf("%s'\n", name? name: "(null)"); }</pre> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | <p>elf(3E), elf32_getshdr(3E), elf32_xlatetof(3E), elf_getdata(3E), attributes(5)</p> | | | | |
| NOTES | <p>A program may call elf_getdata() to retrieve an entire string table section. For some applications, that would be both more efficient and more convenient than using elf_strptr().</p> | | | | |

| | |
|--------------------|--|
| NAME | elf_update – update an ELF descriptor |
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lelf [<i>library ...</i>] #include <libelf.h> off_t elf_update(Elf *elf, Elf_Cmd cmd);</pre> |
| DESCRIPTION | <p>elf_update() causes the library to examine the information associated with an ELF descriptor, <i>elf</i>, and to recalculate the structural data needed to generate the file's image.</p> <p><i>cmd</i> may have the following values:</p> <p>ELF_C_NULL This value tells elf_update() to recalculate various values, updating only the ELF descriptor's memory structures. Any modified structures are flagged with the ELF_F_DIRTY bit. A program thus can update the structural information and then reexamine them without changing the file associated with the ELF descriptor. Because this does not change the file, the ELF descriptor may allow reading, writing, or both reading and writing (see elf_begin (3E)).</p> <p>ELF_C_WRITE If <i>cmd</i> has this value, elf_update() duplicates its ELF_C_NULL actions and also writes any "dirty" information associated with the ELF descriptor to the file. That is, when a program has used elf_getdata(3E) or the elf_flagdata(3E) facilities to supply new (or update existing) information for an ELF descriptor, those data will be examined, coordinated, translated if necessary (see elf32_xlatetof(3E)), and written to the file. When portions of the file are written, any ELF_F_DIRTY bits are reset, indicating those items no longer need to be written to the file (see elf_flagdata(3E)). The sections' data are written in the order of their section header entries, and the section header table is written to the end of the file. When the ELF descriptor was created with elf_begin(), it must have allowed writing the file. That is, the elf_begin() command must have been either ELF_C_RDWR or ELF_C_WRITE.</p> <p>If elf_update() succeeds, it returns the total size of the file image (not the memory image), in bytes. Otherwise an error occurred, and the function returns -1.</p> <p>When updating the internal structures, elf_update() sets some members itself. Members listed below are the application's responsibility and retain the values given by the program.</p> <p>The following table shows ELF Header members:</p> |

| Member | Notes |
|------------------|--|
| e_ident[EL_DATA] | Library controls other e_ident values |
| e_type | |
| e_machine | |
| e_version | |
| e_entry | |
| e_phoff | Only when <code>ELF_F_LAYOUT</code> asserted |
| e_shoff | Only when <code>ELF_F_LAYOUT</code> asserted |
| e_flags | |
| e_shstrndx | |

The following table shows the Program Header members:

| Member | Notes |
|----------|------------------------------|
| p_type | The application controls all |
| p_offset | program header entries |
| p_vaddr | |
| p_paddr | |
| p_filesz | |
| p_memsz | |
| p_flags | |
| p_align | |

The following table shows the Section Header members:

| Member | Notes |
|----------|-------|
| sh_name | |
| sh_type | |
| sh_flags | |
| sh_addr | |

| | |
|--------------|--|
| sh_offset | Only when <code>ELF_F_LAYOUT</code> asserted |
| sh_size | Only when <code>ELF_F_LAYOUT</code> asserted |
| sh_link | |
| sh_info | |
| sh_addralign | Only when <code>ELF_F_LAYOUT</code> asserted |
| sh_entsize | |

The following table shows the Data Descriptor members:

| Member | Notes |
|-----------|--|
| d_buf | |
| d_type | |
| d_size | |
| d_off | Only when <code>ELF_F_LAYOUT</code> asserted |
| d_align | |
| d_version | |

Note that the program is responsible for two particularly important members (among others) in the ELF header. The `e_version` member controls the version of data structures written to the file. If the version is `EV_NONE`, the library uses its own internal version. The `e_ident[EI_DATA]` entry controls the data encoding used in the file. As a special case, the value may be `ELFDATANONE` to request the native data encoding for the host machine. An error occurs in this case if the native encoding doesn't match a file encoding known by the library.

Further note that the program is responsible for the `sh_entsize` section header member. Although the library sets it for sections with known types, it cannot reliably know the correct value for all sections. Consequently, the library relies on the program to provide the values for unknown section types. If the entry size is unknown or not applicable, the value should be set to 0.

When deciding how to build the output file, `elf_update()` obeys the alignments of individual data buffers to create output sections. A section's most strictly aligned data buffer controls the section's alignment. The library also inserts padding between buffers, as necessary, to ensure the proper alignment of each buffer.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`elf(3E)`, `elf32_fsize(3E)`, `elf32_getehdr(3E)`, `elf32_getshdr(3E)`,
`elf32_xlatetof(3E)`, `elf_begin(3E)`, `elf_flagdata(3E)`,
`elf_getdata(3E)`, `attributes(5)`

NOTES

As mentioned above, the `ELF_C_WRITE` command translates data as necessary, before writing them to the file. This translation is *not* always transparent to the application program. If a program has obtained pointers to data associated with a file (for example, see `elf32_getehdr(3E)` and `elf_getdata(3E)`), the program should reestablish the pointers after calling `elf_update()`.

| NAME | elf_version – coordinate ELF library and application versions | | | | |
|--------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lelf [library ...] #include <libelf.h> unsigned elf_version(unsigned ver);</pre> | | | | |
| DESCRIPTION | <p>As elf(3E) explains, the program, the library, and an object file have independent notions of the latest ELF version. elf_version() lets a program query the ELF library's <i>internal version</i>. It further lets the program specify what memory types it uses by giving its own <i>working version</i>, <code>ver</code>, to the library. Every program that uses the ELF library must coordinate versions as described below.</p> <p>The header <code><libelf.h></code> supplies the version to the program with the macro <code>EV_CURRENT</code>. If the library's internal version (the highest version known to the library) is lower than that known by the program itself, the library may lack semantic knowledge assumed by the program. Accordingly, elf_version() will not accept a working version unknown to the library.</p> <p>Passing <code>ver</code> equal to <code>EV_NONE</code> causes elf_version() to return the library's internal version, without altering the working version. If <code>ver</code> is a version known to the library, elf_version() returns the previous (or initial) working version number. Otherwise, the working version remains unchanged and elf_version() returns <code>EV_NONE</code>.</p> | | | | |
| EXAMPLES | <p>EXAMPLE 1 A sample display of using the elf_version() function.</p> <p>The following excerpt from an application program protects itself from using an older library:</p> <pre>if (elf_version(EV_CURRENT) == EV_NONE) { /* library out of date */ /* recover from error */ }</pre> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | elf(3E) , elf32_xlatetof(3E) , elf_begin(3E) , attributes(5) | | | | |

NOTES | The working version should be the same for all operations on a particular ELF descriptor. Changing the version between operations on a descriptor will probably not give the expected results.

| NAME | encrypt – encoding function | | | | |
|----------------------|---|----------------|-----------------|----------|------|
| SYNOPSIS | <pre>#include <unistd.h> void encrypt(char <i>block</i>[64], int <i>edflag</i>);</pre> | | | | |
| DESCRIPTION | <p>The encrypt() function provides (rather primitive) access to the hashing algorithm employed by the crypt(3C) function. The key generated by setkey(3C) is used to encrypt the string <i>block</i> with encrypt().</p> <p>The <i>block</i> argument to encrypt() is an array of length 64 bytes containing only the bytes with numerical value of 0 and 1. The array is modified in place to a similar array using the key set by setkey(3C). If <i>edflag</i> is 0, the argument is encoded. If <i>edflag</i> is 1, the argument may be decoded (see the USAGE section below); if the argument is not decoded, errno will be set to ENOSYS.</p> | | | | |
| RETURN VALUES | The encrypt() function returns no value. | | | | |
| ERRORS | <p>The encrypt() function will fail if:</p> <p>ENOSYS The functionality is not supported on this implementation.</p> | | | | |
| USAGE | <p>In some environments, decoding may not be implemented. This is related to U.S. Government restrictions on encryption and decryption routines: the DES decryption algorithm cannot be exported outside the U.S.A. Historical practice has been to ship a different version of the encryption library without the decryption feature in the routines supplied. Thus the exported version of encrypt() does encoding but not decoding.</p> <p>Because encrypt() does not return a value, applications wishing to check for errors should set errno to 0, call encrypt(), then test errno and, if it is non-zero, assume an error has occurred.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe | | | | |
| SEE ALSO | crypt(3C) , setkey(3C) , attributes(5) | | | | |

| | |
|--------------------|---|
| NAME | end, _end, etext, _etext, edata, _edata – last locations in program |
| SYNOPSIS | extern _etext ; extern _edata ; extern _end ; |
| DESCRIPTION | <p>These names refer neither to routines nor to locations with interesting contents; only their addresses are meaningful.</p> <p><code>_etext</code> The address of <code>_etext</code> is the first location after the program text.</p> <p><code>_edata</code> The address of <code>_edata</code> is the first location after the initialized data region.</p> <p><code>_end</code> The address of <code>_end</code> is the first location after the uninitialized data region.</p> |
| USAGE | <p>When execution begins, the program break (the first location beyond the data) coincides with <code>_end</code>, but the program break may be reset by the <code>brk(2)</code>, <code>malloc(3C)</code>, and the standard input/output library (see <code>stdio(3S)</code>), functions by the profile (<code>-p</code>) option of <code>cc(1B)</code>, and so on. Thus, the current value of the program break should be determined by <code>sbrk ((char *)0)</code>.</p> <p>References to <code>end</code>, <code>etext</code>, and <code>edata</code>, without a preceding underscore will be aliased to the associated symbol that begins with the underscore.</p> |
| SEE ALSO | <code>cc(1B)</code> , <code>brk(2)</code> , <code>malloc(3C)</code> , <code>stdio(3S)</code> |

| | |
|--------------------|---|
| NAME | endhostent, gethostbyaddr, gethostbyname, gethostent, sethostent – network host database functions |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lxnet [<i>library</i> ...] #include <netdb.h> extern int h_errno; void endhostent(void); struct hostent * gethostbyaddr(const void * <i>addr</i>, size_t <i>len</i>, int <i>type</i>); struct hostent * gethostbyname(const char * <i>name</i>); struct hostent * gethostent(void); void sethostent(int <i>stayopen</i>); </pre> |
| DESCRIPTION | <p>The gethostent(), gethostbyaddr(), and gethostbyname() functions each return a pointer to a <code>hostent</code> structure, the members of which contain the fields of an entry in the network host database.</p> <p>The gethostent() function reads the next entry of the database, opening a connection to the database if necessary.</p> <p>The gethostbyaddr() function searches the database and finds an entry which matches the address family specified by the <code>type</code> argument and which matches the address pointed to by the <code>addr</code> argument, opening a connection to the database if necessary. The <code>addr</code> argument is a pointer to the binary-format (that is, not null-terminated) address in network byte order, whose length is specified by the <code>len</code> argument. The datatype of the address depends on the address family. For an address of type <code>AF_INET</code>, this is an <code>in_addr</code> structure, defined in <code><netinet/in.h></code>.</p> <p>The gethostbyname() function searches the database and finds an entry which matches the host name specified by the <code>name</code> argument, opening a connection to the database if necessary. If <code>name</code> is an alias for a valid host name, the function returns information about the host name to which the alias refers, and <code>name</code> is included in the list of aliases returned.</p> |

The **sethostent()** function opens a connection to the network host database, and sets the position of the next entry to the first entry. If the *stayopen* argument is non-zero, the connection to the host database will not be closed after each call to **gethostent()** (either directly, or indirectly through one of the other `gethost*()` functions).

The **endhostent()** function closes the connection to the database.

USAGE

The **gethostent()**, **gethostbyaddr()**, and **gethostbyname()** functions may return pointers to static data, which may be overwritten by subsequent calls to any of these functions.

These functions are generally used with the Internet address family.

RETURN VALUES

On successful completion, **gethostbyaddr()**, **gethostbyname()** and **gethostent()** return a pointer to a `hostent` structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer is returned.

On unsuccessful completion, **gethostbyaddr()** and **gethostbyname()** functions set *h_errno* to indicate the error.

ERRORS

No errors are defined for **endhostent()**, **gethostent()** and **sethostent()**.

The **gethostbyaddr()** and **gethostbyname()** functions will fail in the following cases, setting *h_errno* to the value shown in the list below. Any changes to `errno` are unspecified.

| | |
|----------------|---|
| HOST_NOT_FOUND | No such host is known. |
| NO_DATA | The server recognised the request and the name but no address is available. Another type of request to the name server for the domain might return an answer. |
| NO_RECOVERY | An unexpected server failure occurred which can not be recovered. |
| TRY_AGAIN | A temporary and possibly transient error occurred, such as a failure of a server to respond. |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

endhostent(3XN)

X/Open Networking Services Library Functions

SEE ALSO | `endservent(3XN)`, `htonl(3XN)`, `inet_addr(3XN)`, `attributes(5)`

| | |
|--------------------|--|
| NAME | endnetent, getnetbyaddr, getnetbyname, getnetent, setnetent – network database functions |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lxnet [<i>library</i> ...] #include <netdb.h> void endnetent(void); struct netent *getnetbyaddr(in_addr_t <i>net</i>, int <i>type</i>); struct netent *getnetbyname(const char * <i>name</i>); struct netent *getnetent(void); void setnetent(int <i>stayopen</i>);</pre> |
| DESCRIPTION | <p>The getnetbyaddr(), getnetbyname() and getnetent(), functions each return a pointer to a <code>netent</code> structure, the members of which contain the fields of an entry in the network database.</p> <p>The getnetent() function reads the next entry of the database, opening a connection to the database if necessary.</p> <p>The getnetbyaddr() function searches the database from the beginning, and finds the first entry for which the address family specified by <code>type</code> matches the <code>n_addrtype</code> member and the network number <code>net</code> matches the <code>n_net</code> member, opening a connection to the database if necessary. The <code>net</code> argument is the network number in host byte order.</p> <p>The getnetbyname() function searches the database from the beginning and finds the first entry for which the network name specified by <code>name</code> matches the <code>n_name</code> member, opening a connection to the database if necessary.</p> <p>The setnetent() function opens and rewinds the database. If the <code>stayopen</code> argument is non-zero, the connection to the net database will not be closed after each call to getnetent() (either directly, or indirectly through one of the other <code>getnet*()</code> functions).</p> <p>The endnetent() function closes the database.</p> |

USAGE The **getnetbyaddr()** , **getnetbyname()** and **getnetent()** , functions may return pointers to static data, which may be overwritten by subsequent calls to any of these functions.

These functions are generally used with the Internet address family.

RETURN VALUES On successful completion, **getnetbyaddr()** , **getnetbyname()** and **getnetent()** , return a pointer to a `netent` structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer is returned.

ERRORS No errors are defined.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO **attributes(5)**

| | |
|--------------------|--|
| NAME | endprotoent, getprotobyname, getprotobynumber, getprotoent, setprotoent – network protocol database functions |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lxnet [<i>library</i> ...] #include <netdb.h> void endprotoent(void); struct protoent * getprotobyname(const char * <i>name</i>); struct protoent * getprotobynumber(int <i>proto</i>); struct protoent * getprotoent(void); void setprotoent(int <i>stayopen</i>);</pre> |
| DESCRIPTION | <p>The getprotobyname(), getprotobynumber() and getprotoent(), functions each return a pointer to a <code>protoent</code> structure, the members of which contain the fields of an entry in the network protocol database.</p> <p>The getprotoent() function reads the next entry of the database, opening a connection to the database if necessary.</p> <p>The getprotobyname() function searches the database from the beginning and finds the first entry for which the protocol name specified by <i>name</i> matches the <code>p_name</code> member, opening a connection to the database if necessary.</p> <p>The getprotobynumber() function searches the database from the beginning and finds the first entry for which the protocol number specified by <i>number</i> matches the <code>p_proto</code> member, opening a connection to the database if necessary.</p> <p>The setprotoent() function opens a connection to the database, and sets the next entry to the first entry. If the <i>stayopen</i> argument is non-zero, the connection to the network protocol database will not be closed after each call to getprotoent() (either directly, or indirectly through one of the other <code>getproto*()</code> functions).</p> <p>The endprotoent() function closes the connection to the database.</p> |

USAGE The **getprotobyname()** , **getprotobynumber()** and **getprotoent()** functions may return pointers to static data, which may be overwritten by subsequent calls to any of these functions.

These functions are generally used with the Internet address family.

RETURN VALUES On successful completion, **getprotobyname()** , **getprotobynumber()** and **getprotoent()** functions return a pointer to a `protoent` structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer is returned.

ERRORS No errors are defined.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO **attributes(5)**

| | |
|--------------------|--|
| NAME | endservent, getservbyport, getservbyname, getservent, setservent – network services database functions |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lxnet [<i>library</i> ...] #include <netdb.h> void endservent(void); struct servent * getservbyname(const char * <i>name</i>, const char * <i>proto</i>); struct servent * getservbyport(int <i>port</i>, const char * <i>proto</i>); struct servent * getservent(void); void setservent(int <i>stayopen</i>);</pre> |
| DESCRIPTION | <p>The getservbyname() , getservbyport() and getservent() functions each return a pointer to a <code>servent</code> structure, the members of which contain the fields of an entry in the network services database.</p> <p>The getservent() function reads the next entry of the database, opening a connection to the database if necessary.</p> <p>The getservbyname() function searches the database from the beginning and finds the first entry for which the service name specified by <i>name</i> matches the <code>s_name</code> member and the protocol name specified by <i>proto</i> matches the <code>s_proto</code> member, opening a connection to the database if necessary. If <i>proto</i> is a null pointer, any value of the <code>s_proto</code> member will be matched.</p> <p>The getservbyport() function searches the database from the beginning and finds the first entry for which the port specified by <i>port</i> matches the <code>s_port</code> member and the protocol name specified by <i>proto</i> matches the <code>s_proto</code> member, opening a connection to the database if necessary. If <i>proto</i> is a null pointer, any value of the <code>s_proto</code> member will be matched. The <i>port</i> argument must be in network byte order.</p> <p>The setservent() function opens a connection to the database, and sets the next entry to the first entry. If the <i>stayopen</i> argument is non-zero, the net database</p> |

will not be closed after each call to the **getservent()** function (either directly, or indirectly through one of the other `getserv*()` functions).

The **endservent()** function closes the database.

USAGE

The *port* argument of **getservbyport()** need not be compatible with the port values of all address families.

The **getservent()** , **getservbyname()** and **getservbyport()** functions may return pointers to static data, which may be overwritten by subsequent calls to any of these functions.

These functions are generally used with the Internet address family.

RETURN VALUES

On successful completion, **getservbyname()** , **getservbyport()** and **getservent()** return a pointer to a `servent` structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer is returned.

ERRORS

No errors are defined.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

endhostent(3XN) , **endprotoent(3XN)** , **htonl(3XN)** , **inet_addr(3XN)** , **attributes(5)**

| | |
|----------------------|---|
| NAME | endwin, isendwin – restore initial terminal environment |
| SYNOPSIS | <pre>#include <curses.h> int endwin(void); int isendwin(void);</pre> |
| DESCRIPTION | <p>The endwin() function restores tty modes, resets the terminal, and moves the cursor to the lower left corner of the screen. This function should be called before exiting or escaping X/Open Curses temporarily. To resume X/Open Curses after a temporary escape, call refresh(3XC) or doupdate(3XC) .</p> <p>If the program interacts with multiple terminals, call endwin() for each terminal.</p> <p>The isendwin() function determines whether or not a screen has been refreshed.</p> |
| RETURN VALUES | <p>On success, the endwin() function returns <code>OK</code> . Otherwise, it returns <code>ERR</code> .</p> <p>The isendwin() function returns <code>TRUE</code> if endwin() has been called without subsequent calls to refresh() . Otherwise, it returns <code>FALSE</code> .</p> |
| ERRORS | None. |
| SEE ALSO | doupdate(3XC) |

| | |
|----------------------|---|
| NAME | erasechar, erasewchar, killchar, killwchar – return current ERASE or KILL characters |
| SYNOPSIS | <pre>#include <curses.h> char erasechar(void); int erasewchar(wchar_t * ch); char killchar(void); int killwchar(wchar_t * ch);</pre> |
| PARAMETERS | <p>ch Is a pointer to a location where a character may be stored.</p> |
| DESCRIPTION | <p>The erasechar() function returns the current ERASE character from the tty driver. This character is used to delete the previous character during keyboard input. The returned value can be used when including deletion capability in interactive programs.</p> <p>The killchar() function is similar to erasechar() . It returns the current KILL character.</p> <p>The erasewchar() and killwchar() functions are similar to erasechar() and killchar() respectively, but store the ERASE or KILL character in the object pointed to by <i>ch</i> .</p> |
| RETURN VALUES | <p>For erasechar() and killchar() , the terminal's current ERASE or KILL character is returned.</p> <p>On success, the erasewchar() and killwchar() functions return OK . Otherwise, they return ERR .</p> |
| SEE ALSO | getch(3XC) , getstr(3XC) , get_wch(3XC) |

NAME | erf, erfc – error and complementary error functions

SYNOPSIS | cc [
flag
 ...]
file
 ...
 -lm
 [
library
 ...]
 #include <math.h>

double **erf**(double *x*);
 double **erfc**(double *x*);

DESCRIPTION | The **erf()** function computes the error function of *x*, defined as:


The **erfc()** function computes $1.0 - \text{erf}(x)$.

RETURN VALUES | Upon successful completion, **erf()** and **erfc()** return the value of the error function and complementary error function, respectively.

If *x* is NaN, NaN is returned.

ERRORS | No errors will occur.

USAGE | The **erfc()** function is provided because of the extreme loss of relative accuracy if **erf(x)** is called for large *x* and the result subtracted from 1.0.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **isnan(3M)**, **attributes(5)**

| | |
|--------------------|---|
| NAME | ethers, ether_ntoa, ether_aton, ether_ntohost, ether_hostton, ether_line – Ethernet address mapping operations |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lsocket -lnsl [<i>library</i> ...] #include <sys/types.h> #include <sys/socket.h> #include <net/if.h> #include <netinet/in.h> #include <netinet/if_ether.h> char * ether_ntoa(struct ether_addr * e); struct ether_addr * ether_aton(char * s); int ether_ntohost(char * hostname, struct ether_addr * e); int ether_hostton(char * hostname, struct ether_addr * e); int ether_line(char * l, struct ether_addr * e, char * hostname); </pre> |
| DESCRIPTION | <p>These routines are useful for mapping 48 bit Ethernet numbers to their ASCII representations or their corresponding host names, and vice versa.</p> <p>The function ether_ntoa() converts a 48 bit Ethernet number pointed to by <i>e</i> to its standard ASCII representation; it returns a pointer to the ASCII string. The representation is of the form <i>x : x : x : x : x : x</i> where <i>x</i> is a hexadecimal number between 0 and ff. The function ether_aton() converts an ASCII string in the standard representation back to a 48 bit Ethernet number; the function returns NULL if the string cannot be scanned successfully.</p> <p>The function ether_ntohost() maps an Ethernet number (pointed to by <i>e</i>) to its associated hostname. The string pointed to by <i>hostname</i> must be long enough to hold the hostname and a NULL character. The function returns zero upon success and non-zero upon failure. Inversely, the function ether_hostton() maps a hostname string to its corresponding Ethernet number; the function modifies the Ethernet number pointed to by <i>e</i>. The function also returns zero</p> |

upon success and non-zero upon failure. In order to do the mapping, both these functions may lookup one or more of the following sources: the ethers file, the NIS maps "ethers.byname" and "ethers.byaddr" and the NIS+ table "ethers". The sources and their lookup order are specified in the `/etc/nsswitch.conf` file (see `nsswitch.conf(4)` for details).

The function `ether_line()` scans a line (pointed to by `l`) and sets the hostname and the Ethernet number (pointed to by `e`). The string pointed to by `hostname` must be long enough to hold the hostname and a `NULL` character. The function returns zero upon success and non-zero upon failure. The format of the scanned line is described by `ethers(4)`.

FILES

`/etc/ethers`

`/etc/nsswitch.conf`

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`ethers(4)`, `nsswitch.conf(4)`, `attributes(5)`

BUGS

Programs that call `ether_hostton()` or `ether_ntohost()` routines cannot be linked statically since the implementation of these routines requires dynamic linker functionality to access shared objects at run time.

| NAME | euclen, euccol, eucscol – get byte length and display width of EUC characters | | | | |
|--------------------|---|----------------|-----------------|----------|-------------------------|
| SYNOPSIS | <pre>#include <euc.h> int euclen(const unsigned char * s); int euccol(const unsigned char * s); int eucscol(const unsigned char * str);</pre> | | | | |
| DESCRIPTION | <p>The euclen() function returns the length in bytes of the Extended Unix Code (EUC) character pointed to by <i>s</i>, including single-shift characters, if present.</p> <p>The euccol() function returns the screen column width of the EUC character pointed to by <i>s</i>.</p> <p>The eucscol() function returns the screen column width of the EUC string pointed to by <i>str</i>.</p> <p>For the euclen() and euccol(), functions, <i>s</i> points to the first byte of the character. This byte is examined to determine its codeset. The character type table for the current <i>locale</i> is used for codeset byte length and display width information.</p> | | | | |
| USAGE | <p>These functions will work only with EUC locales.</p> <p>These functions can be used safely in multithreaded applications, as long as setlocale(3C) is not called to change the locale.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" data-bbox="500 1241 1398 1325"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe with exceptions</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe with exceptions |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe with exceptions | | | | |
| SEE ALSO | getwidth(3C) , setlocale(3C) , attributes(5) | | | | |

NAME | exit, _exithandle – terminate process

SYNOPSIS | #include <stdlib.h>

| void **exit**(int *status*);

| void **_exithandle**(void);

DESCRIPTION | The **exit()** function terminates a process by calling first **_exithandle()** and then **_exit()** (see **exit(2)**).

| The **_exithandle()** function calls any functions registered through the **atexit(3C)** function in the reverse order of their registration. This action includes executing all finalization code from the *.fini* sections of all objects that are part of the process.

| The **_exithandle()** function is intended for use *only* with **_exit()** , and allows for specialized processing such as **dldump(3X)** to be performed. Normal process execution should not be continued after a call to **_exithandle()** has occurred, as internal data structures may have been torn down due to **atexit()** or *.fini* processing.

| The symbols **EXIT_SUCCESS** and **EXIT_FAILURE** are defined in the header **<stdlib.h>** and may be used as the value of *status* to indicate successful or unsuccessful termination, respectively.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO | **exit(2)** , **atexit(3C)** , **dldump(3X)** , **attributes(5)**

| NAME | exp – exponential function | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | cc [<i>flag ...</i>] <i>file ...</i> -lm [<i>library ...</i>] #include <math.h> double exp (double <i>x</i>); | | | | |
| DESCRIPTION | The exp() function computes the exponential of <i>x</i> , defined as e^x . | | | | |
| RETURN VALUES | Upon successful completion, exp() returns the exponential of <i>x</i> . If the correct value would cause overflow, exp() returns HUGE_VAL and sets <i>errno</i> to ERANGE. If the correct value would cause underflow to zero, exp() returns 0 and may set <i>errno</i> to ERANGE. If <i>x</i> is NaN, NaN is returned. For exceptional cases, matherr (3M) tabulates the values to be returned as dictated by Standards other than XPG4. | | | | |
| ERRORS | The exp() function will fail if: ERANGE the result overflows. The exp() function may fail if: ERANGE the result underflows. | | | | |
| USAGE | An application wishing to check for error situations should set <i>errno</i> to 0 before calling exp() . If <i>errno</i> is non-zero on return, or the return value is NaN an error has occurred. | | | | |
| ATTRIBUTES | See attributes (5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | isnan (3M), log (3M), matherr (3M), mp (3M), attributes (5), standards (5) | | | | |
| NOTES | Prior to Solaris 2.6, there was a conflict between the pow function in this library and the pow function in the libmp library. This conflict was resolved by prepending mp_ to all functions in the libmp library. See mp (3M) for details. | | | | |

NAME | expm1 – computes exponential functions

SYNOPSIS | cc [*flag ...*] *file ...* -lm [*library ...*]
 | #include <math.h>
 | double expm1(double x);

DESCRIPTION | The **expm1()** function computes $e^x - 1.0$.

RETURN VALUES | If *x* is NaN, then the function returns NaN.
 | If *x* is positive infinity, **expm1()** returns positive infinity.
 | If *x* is negative infinity, **expm1()** returns -1.0 .
 | If the value overflows, **expm1()** returns HUGE_VAL.

ERRORS | No errors will occur.

USAGE | The value of `expm1(x)` may be more accurate than `exp(x) - 1.0` for small values of *x*.
 | The **expm1()** and **log1p(3M)** functions are useful for financial calculations of $((1+x)^n - 1)/x$, namely:
 | `expm1(n * log1p(x)) / x`
 | when *x* is very small (for example, when performing calculations with a small daily interest rate). These functions also simplify writing accurate inverse hyperbolic functions.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **exp(3M)**, **ilogb(3M)**, **log1p(3M)**, **attributes(5)**

NAME fabs – absolute value function

SYNOPSIS cc [*flag ...*] *file ...* -lm [*library ...*]
#include <math.h>
double **fabs**(double *x*);

DESCRIPTION The **fabs()** function computes the absolute value of *x*, $|x|$.

RETURN VALUES Upon successful completion, **fabs()** returns the absolute value of *x*.
If *x* is NaN, NaN is returned.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **isnan(3M)**, **attributes(5)**

| | | | | | | | | | | | | | | | |
|----------------------|--|---------------|---|--------------|---|--------------|---|---------------|---|---------------|---|--------------|---|---------------------|---|
| NAME | fattach – attach a STREAMS-based file descriptor to an object in the file system name space | | | | | | | | | | | | | | |
| SYNOPSIS | <pre>#include <stropts.h> int fattach(int <i>fildev</i>, const char *<i>path</i>);</pre> | | | | | | | | | | | | | | |
| DESCRIPTION | <p>The fattach() function attaches a STREAMS-based file descriptor to an object in the file system name space, effectively associating a name with <i>fildev</i>. <i>fildev</i> must be a valid open file descriptor representing a STREAMS file. <i>path</i> is a path name of an existing object and the user must have appropriate privileges or be the owner of the file and have write permissions. All subsequent operations on <i>path</i> will operate on the STREAMS file until the STREAMS file is detached from the node. <i>fildev</i> can be attached to more than one <i>path</i>, that is, a stream can have several names associated with it.</p> <p>The attributes of the named stream (see stat(2)), are initialized as follows: the permissions, user ID, group ID, and times are set to those of <i>path</i>, the number of links is set to 1, and the size and device identifier are set to those of the streams device associated with <i>fildev</i>. If any attributes of the named stream are subsequently changed (for example, chmod(2)), the attributes of the underlying object are not affected.</p> | | | | | | | | | | | | | | |
| RETURN VALUES | Upon successful completion, fattach() returns 0. Otherwise it returns -1 and sets <i>errno</i> to indicate an error. | | | | | | | | | | | | | | |
| ERRORS | <p>The fattach() function will fail if:</p> <table border="0"> <tr> <td style="padding-right: 20px;">EACCES</td> <td>The user is the owner of <i>path</i> but does not have write permissions on <i>path</i> or <i>fildev</i> is locked.</td> </tr> <tr> <td>EBADF</td> <td>The <i>fildev</i> argument is not a valid open file descriptor.</td> </tr> <tr> <td>EBUSY</td> <td>The <i>path</i> argument is currently a mount point or has a STREAMS file descriptor attached it.</td> </tr> <tr> <td>EINVAL</td> <td>The <i>path</i> argument is a file in a remotely mounted directory.</td> </tr> <tr> <td>EINVAL</td> <td>The <i>fildev</i> argument does not represent a STREAMS file.</td> </tr> <tr> <td>ELOOP</td> <td>Too many symbolic links were encountered in translating <i>path</i>.</td> </tr> <tr> <td>ENAMETOOLONG</td> <td>The size of <i>path</i> exceeds {<i>PATH_MAX</i>}, or the component of a path name is longer than</td> </tr> </table> | EACCES | The user is the owner of <i>path</i> but does not have write permissions on <i>path</i> or <i>fildev</i> is locked. | EBADF | The <i>fildev</i> argument is not a valid open file descriptor. | EBUSY | The <i>path</i> argument is currently a mount point or has a STREAMS file descriptor attached it. | EINVAL | The <i>path</i> argument is a file in a remotely mounted directory. | EINVAL | The <i>fildev</i> argument does not represent a STREAMS file. | ELOOP | Too many symbolic links were encountered in translating <i>path</i> . | ENAMETOOLONG | The size of <i>path</i> exceeds { <i>PATH_MAX</i> }, or the component of a path name is longer than |
| EACCES | The user is the owner of <i>path</i> but does not have write permissions on <i>path</i> or <i>fildev</i> is locked. | | | | | | | | | | | | | | |
| EBADF | The <i>fildev</i> argument is not a valid open file descriptor. | | | | | | | | | | | | | | |
| EBUSY | The <i>path</i> argument is currently a mount point or has a STREAMS file descriptor attached it. | | | | | | | | | | | | | | |
| EINVAL | The <i>path</i> argument is a file in a remotely mounted directory. | | | | | | | | | | | | | | |
| EINVAL | The <i>fildev</i> argument does not represent a STREAMS file. | | | | | | | | | | | | | | |
| ELOOP | Too many symbolic links were encountered in translating <i>path</i> . | | | | | | | | | | | | | | |
| ENAMETOOLONG | The size of <i>path</i> exceeds { <i>PATH_MAX</i> }, or the component of a path name is longer than | | | | | | | | | | | | | | |

{NAME_MAX} while {_POSIX_NO_TRUNC} is in effect.

ENOENT

The *path* argument does not exist.

ENOTDIR

A component of a path prefix is not a directory.

EPERM

The effective user ID is not the owner of *path* or a user with the appropriate privileges.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

fdetach(1M), **chmod(2)**, **mount(2)**, **stat(2)**, **fdetach(3C)**, **isastream(3C)**, **attributes(5)**, **streamio(7I)**

STREAMS Programming Guide

| | |
|--------------------|--|
| NAME | <code>__fbufsize</code> , <code>__flbf</code> , <code>__fpending</code> , <code>__fpurge</code> , <code>__freadable</code> , <code>__freading</code> , <code>__fwritable</code> , <code>__fwriting</code> , <code>_flushlbf</code> - interfaces to stdio FILE structure |
| SYNOPSIS | <pre>#include <stdio.h> #include <stdio_ext.h> size_t __fbufsiz(FILE * stream); int __flbf(FILE * stream); size_t __fpending(FILE * stream); void __fpurge(FILE * stream); int __freadable(FILE * stream); int __freading(FILE * stream); int __fwritable(FILE * stream); int __fwriting(FILE * stream); void _flushlbf(void);</pre> |
| DESCRIPTION | <p>These functions provide portable access to the members of the <code>stdio(3S)</code> FILE structure.</p> <p>The <code>__fbufsize()</code> function returns in bytes the size of the buffer currently in use by the given stream.</p> <p>The <code>__flbf()</code> function returns non-zero if the stream is line-buffered.</p> <p>The <code>__fpending</code> function returns in bytes the amount of output pending on a stream.</p> <p>The <code>__fpurge()</code> function discards any pending buffered I/O on the stream.</p> <p>The <code>__freadable()</code> function returns non-zero if it is possible to read from a stream.</p> <p>The <code>__freading()</code> function returns non-zero if the file is open readonly, or if the last operation on the stream was a read operation such as <code>fread(3S)</code> or <code>fgetc(3S)</code>. Otherwise it returns 0.</p> <p>The <code>__fwritable()</code> function returns non-zero if it is possible to write on a stream.</p> <p>The <code>__fwriting()</code> function returns non-zero if the file is open write-only or append-only, or if the last operation on the stream was a write operation such as <code>fwrite(3S)</code> or <code>fputc(3S)</code>. Otherwise it returns 0.</p> |

The **_flushbf()** function flushes all line-buffered files. It is used when reading from a line-buffered file.

USAGE

Although the contents of the `stdio` FILE structure have always been private to the `stdio` implementation, some applications have needed to obtain information about a `stdio` stream that was not accessible through a supported interface. These applications have resorted to accessing fields of the FILE structure directly, rendering them possibly non-portable to new implementations of `stdio`, or more likely, preventing enhancements to `stdio` that would cause those applications to break.

In the 64-bit environment, the FILE structure is opaque. The functions described here are provided as a means of obtaining the information that up to now has been retrieved directly from the FILE structure. Because they are based on the needs of existing applications (such as `mh` and `emacs`), they may be extended as other programs are ported. Although they may still be non-portable to other operating systems, they will be compatible from each Solaris release to the next. Interfaces that are more portable are under development.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---------------------|-----------------|
| MT-Level | MT-Safe |
| Interface Stability | Evolving |

SEE ALSO

`fgetc(3S)`, `fputc(3S)`, `fread(3S)`, `fwrite(3S)`, `stdio(3S)`, `attributes(5)`

| | |
|----------------------|--|
| NAME | fclose – close a stream |
| SYNOPSIS | <pre>#include <stdio.h> int fclose(FILE *<i>stream</i>);</pre> |
| DESCRIPTION | <p>The fclose() function causes the stream pointed to by <i>stream</i> to be flushed and the associated file to be closed. Any unwritten buffered data for the stream is written to the file; any unread buffered data is discarded. The stream is disassociated from the file. If the associated buffer was automatically allocated, it is deallocated.</p> <p>The fclose() function marks for update the <code>st_ctime</code> and <code>st_mtime</code> fields of the underlying file if the stream is writable and if buffered data has not yet been written to the file. It will perform a <code>close(2)</code> operation on the file descriptor that is associated with the stream pointed to by <i>stream</i>.</p> <p>After the call to fclose(), any use of <i>stream</i> causes undefined behavior.</p> <p>The fclose() function is performed automatically for all open files upon calling <code>exit(2)</code>.</p> |
| RETURN VALUES | Upon successful completion, fclose() returns 0. Otherwise, it returns EOF and sets <code>errno</code> to indicate the error. |
| ERRORS | <p>The fclose() function will fail if:</p> <p>EAGAIN The <code>O_NONBLOCK</code> flag is set for the file descriptor underlying <i>stream</i> and the process would be delayed in the write operation.</p> <p>EBADF The file descriptor underlying stream is not valid.</p> <p>EFBIG An attempt was made to write a file that exceeds the maximum file size or the process's file size limit; or the file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.</p> <p>EINTR The fclose() function was interrupted by a signal.</p> <p>EIO The process is a member of a background process group attempting to write to its controlling terminal, <code>TOSTOP</code> is set, the process is neither ignoring nor blocking <code>SIGTTOU</code> and the process group of the process is orphaned.</p> <p>ENOSPC There was no free space remaining on the device containing the file.</p> |

EPIPE An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A `SIGPIPE` signal will also be sent to the process.

The `fclose()` function may fail if:

ENXIO A request was made of a non-existent device, or the request was beyond the limits of the device.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`close(2)`, `exit(2)`, `getrlimit(2)`, `ulimit(2)`, `fopen(3S)`, `stdio(3S)`, `attributes(5)`

| NAME | fdatasync – synchronize a file's data | | | | |
|----------------------|---|----------------|-----------------|----------|-------------------|
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <unistd.h> int fdatasync(int <i>fildes</i>);</pre> | | | | |
| DESCRIPTION | <p>The fdatasync() function forces all currently queued I/O operations associated with the file indicated by file descriptor <i>fildes</i> to the synchronized I/O completion state.</p> <p>The functionality is as described for fsync(3C) (with the symbol <code>_XOPEN_REALTIME</code> defined), with the exception that all I/O operations are completed as defined for synchronised I/O data integrity completion.</p> | | | | |
| RETURN VALUES | <p>If successful, the fdatasync() function returns 0. Otherwise, the function returns -1 and sets <code>errno</code> to indicate the error. If the fdatasync() function fails, outstanding I/O operations are not guaranteed to have been completed.</p> | | | | |
| ERRORS | <p>The fdatasync() function will fail if:</p> <p>EBADF The <i>fildes</i> argument is not a valid file descriptor open for writing.</p> <p>EINVAL The system does not support synchronized I/O for this file.</p> <p>ENOSYS The function fdatasync() is not supported by the system. In the event that any of the queued I/O operations fail, fdatasync() returns the error conditions defined for read(2) and write(2).</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Async-Signal-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Async-Signal-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Async-Signal-Safe | | | | |
| SEE ALSO | <p>fcntl(2), open(2), read(2), write(2), fsync(3C), aio_fsync(3R), attributes(5), fcntl(5)</p> | | | | |

| | |
|----------------------|---|
| NAME | fdetach – detach a name from a STREAMS-based file descriptor |
| SYNOPSIS | <pre>#include <stropts.h> int fdetach(const char *path);</pre> |
| DESCRIPTION | <p>The fdetach() function detaches a STREAMS-based file from the file to which it was attached by a previous call to fattach(3C). The <i>path</i> argument points to the pathname of the attached STREAMS file. The process must have appropriate privileges or be the owner of the file. A successful call to fdetach() causes all pathnames that named the attached STREAMS file to again name the file to which the STREAMS file was attached. All subsequent operations on <i>path</i> will operate on the underlying file and not on the STREAMS file.</p> <p>All open file descriptions established while the STREAMS file was attached to the file referenced by <i>path</i>, will still refer to the STREAMS file after the fdetach() has taken effect.</p> <p>If there are no open file descriptors or other references to the STREAMS file, then a successful call to fdetach() has the same effect as performing the last close(2) on the attached file.</p> |
| RETURN VALUES | Upon successful completion, fdetach() returns 0. Otherwise, it returns -1 and sets errno to indicate the error. |
| ERRORS | <p>The fdetach() function will fail if:</p> <p>EACCES Search permission is denied on a component of the path prefix.</p> <p>EPERM The effective user ID is not the owner of <i>path</i> and the process does not have appropriate privileges.</p> <p>ENOTDIR A component of the path prefix is not a directory.</p> <p>ENOENT A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.</p> <p>EINVAL The <i>path</i> argument names a file that is not currently attached.</p> <p>ENAMETOOLONG The size of a pathname exceeds PATH_MAX, or a pathname component is longer than NAME_MAX while _POSIX_NO_TRUNC is in effect.</p> <p>ELOOP Too many symbolic links were encountered in resolving <i>path</i>.</p> <p>The fdetach() function may fail if:</p> |

ENAMETOOLONG Pathname resolution of a symbolic link produced an intermediate result whose length exceeds PATH_MAX.

SEE ALSO

fdetach(1M), **close(2)**, **fattach(3C)**, **streamio(7I)**
STREAMS Programming Guide

| | | | | | | | | | | | | | |
|----------------------|--|--------------|--|---------------|---|---------|---|------------------|---|------------------|---|------------------|--|
| NAME | fdopen – associate a stream with a file descriptor | | | | | | | | | | | | |
| SYNOPSIS | <pre>#include <stdio.h> FILE *fdopen(int <i>fdes</i>, const char *<i>mode</i>);</pre> | | | | | | | | | | | | |
| DESCRIPTION | <p>The fdopen() function associates a stream with a file descriptor <i>fdes</i>.</p> <p>The <i>mode</i> argument is a character string having one of the following values:</p> <table border="0" style="margin-left: 20px;"> <tr> <td>r or rb</td> <td>Open a file for reading.</td> </tr> <tr> <td>w or wb</td> <td>Open a file for writing.</td> </tr> <tr> <td>a or ab</td> <td>Open a file for writing at end of file.</td> </tr> <tr> <td>r+ or rb+ or r+b</td> <td>Open a file for update (reading and writing).</td> </tr> <tr> <td>w+ or wb+ or w+b</td> <td>Open a file for update (reading and writing).</td> </tr> <tr> <td>a+ or ab+ or a+b</td> <td>Open a file for update (reading and writing) at end of file.</td> </tr> </table> <p>The meaning of these flags is exactly as specified for the fopen(3S) function, except that modes beginning with <i>w</i> do not cause truncation of the file.</p> <p>The mode of the stream must be allowed by the file access mode of the open file. The file position indicator associated with the new stream is set to the position indicated by the file offset associated with the file descriptor.</p> <p>The fdopen() function preserves the offset maximum previously set for the open file description corresponding to <i>fdes</i>.</p> <p>The error and end-of-file indicators for the stream are cleared. The fdopen() function may cause the <code>st_atime</code> field of the underlying file to be marked for update.</p> <p>If <i>fdes</i> refers to a shared memory object, the result of the fdopen() function is unspecified.</p> | r or rb | Open a file for reading. | w or wb | Open a file for writing. | a or ab | Open a file for writing at end of file. | r+ or rb+ or r+b | Open a file for update (reading and writing). | w+ or wb+ or w+b | Open a file for update (reading and writing). | a+ or ab+ or a+b | Open a file for update (reading and writing) at end of file. |
| r or rb | Open a file for reading. | | | | | | | | | | | | |
| w or wb | Open a file for writing. | | | | | | | | | | | | |
| a or ab | Open a file for writing at end of file. | | | | | | | | | | | | |
| r+ or rb+ or r+b | Open a file for update (reading and writing). | | | | | | | | | | | | |
| w+ or wb+ or w+b | Open a file for update (reading and writing). | | | | | | | | | | | | |
| a+ or ab+ or a+b | Open a file for update (reading and writing) at end of file. | | | | | | | | | | | | |
| RETURN VALUES | <p>Upon successful completion, fdopen() returns a pointer to a stream. Otherwise, a null pointer is returned and <code>errno</code> is set to indicate the error.</p> <p>The fdopen() function may fail and not set <code>errno</code> if there are no free <code>stdio</code> streams.</p> | | | | | | | | | | | | |
| ERRORS | <p>The fdopen() function may fail if:</p> <table border="0" style="margin-left: 20px;"> <tr> <td>EBADF</td> <td>The <i>fdes</i> argument is not a valid file descriptor.</td> </tr> <tr> <td>EINVAL</td> <td>The <i>mode</i> argument is not a valid mode.</td> </tr> </table> | EBADF | The <i>fdes</i> argument is not a valid file descriptor. | EINVAL | The <i>mode</i> argument is not a valid mode. | | | | | | | | |
| EBADF | The <i>fdes</i> argument is not a valid file descriptor. | | | | | | | | | | | | |
| EINVAL | The <i>mode</i> argument is not a valid mode. | | | | | | | | | | | | |

EMFILE The number of streams currently open in the calling process is either `FOPEN_MAX` or `STREAM_MAX`.

ENOMEM Insufficient space to allocate a buffer.

USAGE

The number of streams that a process can have open at one time is `STREAM_MAX`. If defined, it has the same value as `FOPEN_MAX`.

File descriptors are obtained from calls like `open(2)`, `dup(2)`, `creat(2)` or `pipe(2)`, which open files but do not return streams. Streams are necessary input for almost all of the Section 3S library routines.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`creat(2)`, `dup(2)`, `open(2)`, `pipe(2)`, `fclose(3S)`, `fopen(3S)`, `attributes(5)`

| NAME | ferror, feof, clearerr, fileno – stream status inquiries | | | | |
|--------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <stdio.h> int ferror(FILE * stream); int feof(FILE * stream); void clearerr(FILE * stream); int fileno(FILE * stream);</pre> | | | | |
| DESCRIPTION | <p>The ferror() function returns a non-zero value when an error has previously occurred reading from or writing to the named <i>stream</i> (see intro(3)). It returns 0 otherwise.</p> <p>The feof() function returns a non-zero value when EOF has previously been detected reading the named input <i>stream</i>. It returns 0 otherwise.</p> <p>The clearerr() function resets the error indicator and EOF indicator to 0 on the named <i>stream</i>.</p> <p>The fileno() function returns the integer file descriptor associated with the named <i>stream</i>; see open(2).</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" data-bbox="496 1115 1395 1203"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | open(2) , intro(3) , fopen(3S) , stdio(3S) , attributes(5) | | | | |

| | |
|----------------------|--|
| NAME | fflush – flush a stream |
| SYNOPSIS | <pre>#include <stdio.h> int fflush(FILE *stream);</pre> |
| DESCRIPTION | <p>If <i>stream</i> points to an output stream or an update stream in which the most recent operation was not input, fflush() causes any unwritten data for that stream to be written to the file, and the <i>st_ctime</i> and <i>st_mtime</i> fields of the underlying file are marked for update.</p> <p>If <i>stream</i> is a null pointer, fflush() performs this flushing action on all streams for which the behavior is defined above.</p> |
| RETURN VALUES | Upon successful completion, fflush() returns 0. Otherwise, it returns EOF and sets <i>errno</i> to indicate the error. |
| ERRORS | <p>The fflush() function will fail if:</p> <p>EAGAIN The <i>O_NONBLOCK</i> flag is set for the file descriptor underlying <i>stream</i> and the process would be delayed in the write operation.</p> <p>EBADF The file descriptor underlying <i>stream</i> is not valid.</p> <p>EFBIG An attempt was made to write a file that exceeds the maximum file size or the process's file size limit; or the file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.</p> <p>EINTR The fflush() function was interrupted by a signal.</p> <p>EIO The process is a member of a background process group attempting to write to its controlling terminal, <i>TOSTOP</i> is set, the process is neither ignoring nor blocking <i>SIGTTOU</i>, and the process group of the process is orphaned.</p> <p>ENOSPC There was no free space remaining on the device containing the file.</p> <p>EPIPE An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A <i>SIGPIPE</i> signal will also be sent to the process.</p> <p>The fflush() function may fail if:</p> <p>ENXIO A request was made of a non-existent device, or the request was beyond the limits of the device.</p> |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`getrlimit(2)`, `ulimit(2)`, `attributes(5)`

| NAME | ffs – find first set bit | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | #include <strings.h> int ffs(const int i); | | | | |
| DESCRIPTION | The ffs() function finds the first bit set (beginning with the least significant bit) and returns the index of that bit. Bits are numbered starting at one (the least significant bit). | | | | |
| RETURN VALUES | The ffs() function returns the index of the first bit set. If <i>i</i> is 0, then ffs() returns 0. | | | | |
| ERRORS | No errors are defined. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | attributes(5) | | | | |

| | |
|---|--|
| NAME | fgetc, getc, getc_unlocked, getchar, getchar_unlocked, getw – get a byte from a stream |
| SYNOPSIS | <pre>#include <stdio.h> int fgetc(FILE * stream); int getc(FILE * stream); int getc_unlocked(FILE * stream); int getchar(void); int getchar_unlocked(void); int getw(FILE * stream);</pre> |
| DESCRIPTION | |
| fgetc() | <p>The fgetc() function obtains the next byte (if present) as an unsigned char converted to an int, from the input stream pointed to by <i>stream</i>, and advances the associated file position indicator for the stream (if defined).</p> <p>The fgetc() function may mark the <code>st_atime</code> field of the file associated with <i>stream</i> for update. The <code>st_atime</code> field will be marked for update by the first successful execution of fgetc(), fgets(3S), fgetwc(3S), fgetws(3S), fread(3S), fscanf(3S), getc(), getchar(), gets(3S) or scanf(3S) using <i>stream</i> that returns data not supplied by a prior call to ungetc(3S) or ungetwc(3S).</p> |
| getc() | The getc() routine is functionally identical to fgetc() , except that it is implemented as a macro. It runs faster than fgetc() , but it takes up more space per invocation and its name cannot be passed as an argument to a function call. |
| getchar() | The getchar() routine is equivalent to <code>getc(stdin)</code> . It is implemented as a macro. |
| getc_unlocked() and getchar_unlocked() | The getc_unlocked() and getchar_unlocked() routines are variants of getc() and getchar() , respectively, that do not lock the stream. It is the caller's responsibility to acquire the stream lock before calling these routines and releasing the lock afterwards; see flockfile(3S) and stdio(3S) . These routines are implemented as macros. |
| getw() | The getw() function reads the next word from the <i>stream</i> . The size of a word is the size of an int and may vary from environment to environment. The getw() function presumes no special alignment in the file. |

The **getw()** function may mark the `st_atime` field of the file associated with *stream* for update. The `st_atime` field will be marked for update by the first successful execution of **fgetc()**, **fgets(3S)**, **fread(3S)**, **getc()**, **getchar()**, **gets(3S)**, **fscanf(3S)** or **scanf(3S)** using *stream* that returns data not supplied by a prior call to **ungetc(3S)**.

RETURN VALUES

Upon successful completion, **fgetc()**, **getc()**, **getc_unlocked()**, **getchar()**, **getchar_unlocked()**, and **getw()** return the next byte from the input stream pointed to by *stream*. If the stream is at end-of-file, the end-of-file indicator for the stream is set and these functions return EOF. If a read error occurs, the error indicator for the stream is set, EOF is returned, and `errno` is set to indicate the error.

ERRORS

The **fgetc()**, **getc()**, **getc_unlocked()**, **getchar()**, **getchar_unlocked()**, and **getw()** functions will fail if data needs to be read and:

EAGAIN The `O_NONBLOCK` flag is set for the file descriptor underlying *stream* and the process would be delayed in the **fgetc()** operation.

EBADF The file descriptor underlying *stream* is not a valid file descriptor open for reading.

EINTR The read operation was terminated due to the receipt of a signal, and no data was transferred.

EIO A physical I/O error has occurred, or the process is in a background process group attempting to read from its controlling terminal, and either the process is ignoring or blocking the `SIGTTIN` signal or the process group is orphaned. This error may also be generated for implementation-dependent reasons.

EOVERFLOW The file is a regular file and an attempt was made to read at or beyond the offset maximum associated with the corresponding stream.

The **fgetc()**, **getc()**, **getc_unlocked()**, **getchar()**, **getchar_unlocked()**, and **getw()** functions may fail if:

ENOMEM Insufficient storage space is available.

ENXIO A request was made of a non-existent device, or the request was outside the capabilities of the device.

USAGE

If the integer value returned by **fgetc()**, **getc()**, **getc_unlocked()**, **getchar()**, **getchar_unlocked()**, and **getw()** is stored into a variable of type `char` and then compared against the integer constant EOF, the comparison may never

succeed, because sign-extension of a variable of type `char` on widening to integer is implementation-dependent.

The `ferror(3S)` or `feof(3S)` functions must be used to distinguish between an error condition and an end-of-file condition.

Functions exist for the `getc()`, `getc_unlocked()`, `getchar()`, and `getchar_unlocked()` macros. To get the function form, the macro name must be undefined (for example, `#undef getc`).

When the macro forms are used, `getc()` and `getc_unlocked()` evaluate the *stream* argument more than once. In particular, `getc(*f++)`; does not work sensibly. The `fgetc()` function should be used instead when evaluating the *stream* argument has side effects.

Because of possible differences in word length and byte ordering, files written using `getw()` are machine-dependent, and may not be read using `getw()` on a different processor.

The `getw()` function is inherently byte stream-oriented and is not tenable in the context of either multibyte character streams or wide-character streams. Application programmers are recommended to use one of the character-based input functions instead.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|------------------|
| MT-Level | See NOTES below. |

SEE ALSO

`intro(3)`, `fclose(3S)`, `feof(3S)`, `fgets(3S)`, `fgetwc(3S)`, `fgetws(3S)`, `flockfile(3S)`, `fopen(3S)`, `fread(3S)`, `fscanf(3S)`, `gets(3S)`, `putc(3S)`, `scanf(3S)`, `stdio(3S)`, `ungetc(3S)`, `ungetwc(3S)`, `attributes(5)`

NOTES

The `fgetc()`, `getc()`, `getchar()`, and `getw()` routines are MT-Safe in multithreaded applications. The `getc_unlocked()` and `getchar_unlocked()` routines are unsafe in multithreaded applications.

| | |
|----------------------|--|
| NAME | fgetpos – get current file position information |
| SYNOPSIS | <pre>#include <stdio.h> int fgetpos(FILE *stream, fpos_t *pos);</pre> |
| DESCRIPTION | The fgetpos() function stores the current value of the file position indicator for the stream pointed to by <i>stream</i> in the object pointed to by <i>pos</i> . The value stored contains unspecified information usable by fsetpos(3S) for repositioning the stream to its position at the time of the call to fgetpos() . |
| RETURN VALUES | Upon successful completion, fgetpos() returns 0. Otherwise, it returns a non-zero value and sets <code>errno</code> to indicate the error. |
| ERRORS | The fgetpos() function may fail if: EBADF The file descriptor underlying <i>stream</i> is not valid. ESPIPE The file descriptor underlying <i>stream</i> is associated with a pipe, a FIFO, or a socket. E_OVERFLOW The current value of the file position cannot be represented correctly in an object of type <code>fpos_t</code> . |
| USAGE | The fgetpos() function has a transitional interface for 64-bit file offsets. See 1f64(5) . |
| SEE ALSO | fopen(3S) , fsetpos(3S) , ftell(3S) , rewind(3S) , ungetc(3S) , 1f64(5) |

| | |
|----------------------|---|
| NAME | fgetwc – get a wide-character code from a stream |
| SYNOPSIS | <pre>#include <stdio.h> #include <wchar.h> wint_t fgetwc(FILE*stream);</pre> |
| DESCRIPTION | <p>The fgetwc() function obtains the next character (if present) from the input stream pointed to by <i>stream</i>, converts that to the corresponding wide-character code and advances the associated file position indicator for the stream (if defined).</p> <p>If an error occurs, the resulting value of the file position indicator for the stream is indeterminate.</p> <p>The fgetwc() function may mark the <code>st_atime</code> field of the file associated with <i>stream</i> for update. The <code>st_atime</code> field will be marked for update by the first successful execution of fgetwc(), fgetc(3S), fgets(3S), fgetws(3S), fread(3S), fscanf(3S), getc(3S), getchar(3S), gets(3S), or scanf(3S) using <i>stream</i> that returns data not supplied by a prior call to ungetc(3S) or ungetwc(3S).</p> |
| RETURN VALUES | <p>Upon successful completion the fgetwc() function returns the wide-character code of the character read from the input stream pointed to by <i>stream</i> converted to a type <code>wint_t</code>.</p> <p>If the stream is at end-of-file, the end-of-file indicator for the stream is set and fgetwc() returns <code>WEOF</code>.</p> <p>If a read error occurs, the error indicator for the stream is set, fgetwc() returns <code>WEOF</code> and sets <code>errno</code> to indicate the error.</p> |
| ERRORS | <p>The fgetwc() function will fail if data needs to be read and:</p> <p>EAGAIN The <code>O_NONBLOCK</code> flag is set for the file descriptor underlying <i>stream</i> and the process would be delayed in the fgetwc() operation.</p> <p>EBADF The file descriptor underlying <i>stream</i> is not a valid file descriptor open for reading.</p> <p>EINTR The read operation was terminated due to the receipt of a signal, and no data was transferred.</p> <p>EIO A physical I/O error has occurred, or the process is in a background process group attempting to read from its controlling terminal and either the process is ignoring or blocking the <code>SIGTTIN</code> signal or the process group is orphaned.</p> |

EOVERFLOW The file is a regular file and an attempt was made to read at or beyond the offset maximum associated with the corresponding *stream*.

The **fgetwc()** function may fail if:

ENOMEM Insufficient storage space is available.

ENXIO A request was made of a non-existent device, or the request was outside the capabilities of the device.

EILSEQ The data obtained from the input stream does not form a valid character.

USAGE

The **ferror(3S)** or **feof(3S)** functions must be used to distinguish between an error condition and an end-of-file condition.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |
| CSI | Enabled |

SEE ALSO

feof(3S), **ferror(3S)**, **fgetc(3S)**, **fgets(3S)**, **fgetws(3S)**, **fopen(3S)**, **fread(3S)**, **fscanf(3S)**, **getc(3S)**, **getchar(3S)**, **gets(3S)**, **scanf(3S)**, **setlocale(3C)**, **ungetc(3S)**, **ungetwc(3S)**, **attributes(5)**

| | |
|----------------------|--|
| NAME | filter – disable use of certain terminal capabilities |
| SYNOPSIS | <pre>#include <curses.h> void filter(void);</pre> |
| DESCRIPTION | <p>The filter() function changes how X/Open Curses initializes terminal capabilities that assume the terminal has more than one line. After a call to filter(), the initscr(3XC) or newterm(3XC) functions also:</p> <ul style="list-style-type: none">■ disable use of <code>clear</code>, <code>cucl</code>, <code>cuul</code> and <code>vpa</code>■ set home string to the value of <code>cr</code>■ set lines to 1 |
| RETURN VALUES | The filter() function does not return a value. |
| ERRORS | None. |
| SEE ALSO | initscr(3XC) , newterm(3XC) |

| | |
|--------------------|--|
| NAME | floating_to_decimal, single_to_decimal, double_to_decimal, extended_to_decimal, quadruple_to_decimal – convert floating-point value to decimal record |
| SYNOPSIS | <pre>#include <floatingpoint.h> void single_to_decimal(single * px, decimal_mode * pm, decimal_record * pd, fp_exception_field_type * ps); void double_to_decimal(double * px, decimal_mode * pm, decimal_record * pd, fp_exception_field_type * ps); void extended_to_decimal(extended * px, decimal_mode * pm, decimal_record * pd, fp_exception_field_type * ps); void quadruple_to_decimal(quadruple * px, decimal_mode * pm, decimal_record * pd, fp_exception_field_type * ps);</pre> |
| DESCRIPTION | <p>The floating_to_decimal() functions convert the floating-point value at <i>*px</i> into a decimal record at <i>*pd</i>, observing the modes specified in <i>*pm</i> and setting exceptions in <i>*ps</i>. If there are no IEEE exceptions, <i>*ps</i> will be zero.</p> <p>If <i>*px</i> is zero, infinity, or NaN, then only <i>pd->sign</i> and <i>pd->fpclass</i> are set. Otherwise <i>pd->exponent</i> and <i>pd->ds</i> are also set so that</p> <pre>(sig)*(pd->ds)*10**(pd->exponent)</pre> <p>is a correctly rounded approximation to <i>*px</i>, where <i>sig</i> is +1 or -1, depending upon whether <i>pd->sign</i> is 0 or -1. <i>pd->ds</i> has at least one and no more than <code>DECIMAL_STRING_LENGTH-1</code> significant digits because one character is used to terminate the string with a NULL.</p> <p><i>pd->ds</i> is correctly rounded according to the IEEE rounding modes in <i>pm->rd</i>. <i>*ps</i> has <i>fp_inexact</i> set if the result was inexact, and has <i>fp_overflow</i> set if the string result does not fit in <i>pd->ds</i> because of the limitation <code>DECIMAL_STRING_LENGTH</code>.</p> <p>If <i>pm->df</i> == <i>floating_form</i>, then <i>pd->ds</i> always contains <i>pm->ndigits</i> significant digits. Thus if <i>*px</i> == 12.34 and <i>pm->ndigits</i> == 8, then <i>pd->ds</i> will contain 12340000 and <i>pd->exponent</i> will contain -6.</p> <p>If <i>pm->df</i> == <i>fixed_form</i> and <i>pm->ndigits</i> >= 0, then <i>pd->ds</i> always contains <i>pm->ndigits</i> after the point and as many digits as necessary before the point. Since the latter is not known in advance, the total number of digits required is returned in <i>pd->ndigits</i>; if that number >= <code>DECIMAL_STRING_LENGTH</code>, then <i>ds</i> is undefined. <i>pd->exponent</i> always gets <i>-pm->ndigits</i>. Thus if <i>*px</i> == 12.34</p> |

and *pm->ndigits* == 1, then *pd->ds* gets 123, *pd->exponent* gets -1, and *pd->ndigits* gets 3.

If *pm->df* == *fixed_form* and *pm->ndigits* < 0, then *pd->ds* always contains *-pm->ndigits* trailing zeros; in other words, rounding occurs *-pm->ndigits* to the left of the decimal point, but the digits rounded away are retained as zeros. The total number of digits required is in *pd->ndigits*. *pd->exponent* always gets 0. Thus if **px* == 12.34 and *pm->ndigits* == -1, then *pd->ds* gets 10, *pd->exponent* gets 0, and *pd->ndigits* gets 2.

pd->more is not used.

`econvert(3)`, `fconvert(3)`, `gconvert(3)`, `printf(3S)`, and `sprintf(3S)` all use `double_to_decimal()`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`econvert(3)`, `fconvert(3)`, `gconvert(3)`, `printf(3S)`, `sprintf(3S)`, `attributes(5)`

| | |
|----------------------|---|
| NAME | flock – apply or remove an advisory lock on an open file |
| SYNOPSIS | <pre> /usr/ucb/cc[<i>flag ...</i>] <i>file ...</i> #include <sys/file.h> int flock(<i>fd</i>, <i>operation</i>); int <i>fd</i>, <i>operation</i>; </pre> |
| DESCRIPTION | <p>flock() applies or removes an <i>advisory</i> lock on the file associated with the file descriptor <i>fd</i>. The compatibility version of flock() has been implemented on top of fcntl(2) locking. It does not provide complete binary compatibility.</p> <p>Advisory locks allow cooperating processes to perform consistent operations on files, but do not guarantee exclusive access (that is, processes may still access files without using advisory locks, possibly resulting in inconsistencies).</p> <p>The locking mechanism allows two types of locks: shared locks and exclusive locks. More than one process may hold a shared lock for a file at any given time, but multiple exclusive, or both shared and exclusive, locks may not exist simultaneously on a file.</p> <p>A lock is applied by specifying an <i>operation</i> parameter LOCK_SH for a shared lock or LOCK_EX for an exclusive lock. The <i>operation</i> parameter may be ORed with LOCK_NB to make the operation non-blocking. To unlock an existing lock, the <i>operation</i> should be LOCK_UN.</p> <p>Read permission is required on a file to obtain a shared lock, and write permission is required to obtain an exclusive lock. Locking a segment that is already locked by the calling process causes the old lock type to be removed and the new lock type to take effect.</p> <p>Requesting a lock on an object that is already locked normally causes the caller to block until the lock may be acquired. If LOCK_NB is included in <i>operation</i>, then this will not happen; instead, the call will fail and the error EWOULDBLOCK will be returned.</p> |
| RETURN VALUES | <p>flock() returns:</p> <p>0 on success.</p> <p>-1 on failure and sets errno to indicate the error.</p> |
| ERRORS | <p>EBADF The argument <i>fd</i> is an invalid descriptor.</p> <p>EINVAL <i>operation</i> is not a valid argument.</p> <p>EOPNOTSUPP The argument <i>fd</i> refers to an object other than a file.</p> |

EWOULDBLOCK The file is locked and the `LOCK_NB` option was specified.

SEE ALSO `lockd(1M)`, `chmod(2)`, `close(2)`, `dup(2)`, `exec(2)`, `fcntl(2)`, `fork(2)`, `open(2)`, `lockf(3C)`

NOTES Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

Locks are on files, not file descriptors. That is, file descriptors duplicated through `dup(2)` or `fork(2)` do not result in multiple instances of a lock, but rather multiple references to a single lock. If a process holding a lock on a file forks and the child explicitly unlocks the file, the parent will lose its lock. Locks are not inherited by a child process.

Processes blocked awaiting a lock may be awakened by signals.

Mandatory locking may occur, depending on the mode bits of the file. See `chmod(2)`.

Locks obtained through the **flock()** mechanism under SunOS 4.1 were known only within the system on which they were placed. This is no longer true.

| | |
|----------------------|--|
| NAME | flockfile, funlockfile, ftrylockfile – acquire and release stream lock |
| SYNOPSIS | <pre>#include <stdio.h> void flockfile(FILE * stream); void funlockfile(FILE * stream); int ftrylockfile(FILE * stream);</pre> |
| DESCRIPTION | <p>The flockfile() function acquires an internal lock of a stream <i>stream</i> . If the lock is already acquired by another thread, the thread calling flockfile() is suspended until it can acquire the lock. In the case that the stream lock is available, flockfile() not only acquires the lock, but keeps track of the number of times it is being called by the current thread. This implies that the stream lock can be acquired more than once by the same thread.</p> <p>The funlockfile() function releases the lock being held by the current thread. In the case of recursive locking, this function must be called the same number of times flockfile() was called. After the number of funlockfile() calls is equal to the number of flockfile() calls, the stream lock is available for other threads to acquire.</p> <p>The ftrylockfile() function acquires an internal lock of a stream <i>stream</i> , only if that object is available. In essence ftrylockfile() is a non-blocking version of flockfile() .</p> |
| RETURN VALUES | The ftrylockfile() function returns 0 on success and non-zero to indicate a lock cannot be acquired. |
| EXAMPLES | <p>EXAMPLE 1 A sample program of flockfile() .</p> <p>The following example prints everything out together, blocking other threads that might want to write to the same file between calls to fprintf(3S) :</p> <pre>FILE iop; flockfile(iop); fprintf(iop, "hello "); fprintf(iop, "world"); fputc(iop, 'a'); funlockfile(iop);</pre> <p>An unlocked interface is available in case performance is an issue. For example:</p> <pre>flockfile(iop); while (!feof(iop)) { *c++ = getc_unlocked(iop); } funlockfile(iop);</pre> |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

intro(3), **ferror(3S)**, **fprintf(3S)**, **getc(3S)**, **putc(3S)**, **stdio(3S)**, **ungetc(3S)**, **attributes(5)**, **standards(5)**

NOTES

The interfaces on this page are as specified in IEEE Std 1003.1c. See **standards(5)**.

NAME floor – floor function

SYNOPSIS cc [*flag ...*] *file ...* -lm [*library ...*]
#include <math.h>
double **floor**(double *x*);

DESCRIPTION The **floor()** function computes the largest integral value not greater than *x*.

RETURN VALUES Upon successful completion, **floor()** returns the largest integral value not greater than *x*, expressed as a `double`.
If *x* is NaN, NaN is returned.
If *x* is ±Inf or ±0, *x* is returned.

ERRORS No errors will occur.

USAGE The integral value returned by **floor()** as a `double` might not be expressible as an `int` or `long int`. The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **ceil(3M)**, **isnan(3M)**, **attributes(5)**

| | |
|----------------------|---|
| NAME | flushinp – discard type-ahead characters |
| SYNOPSIS | <pre>#include <curses.h> int flushinp(void);</pre> |
| DESCRIPTION | The flushinp() function discards all type-ahead characters (characters typed by the user, but not yet processed by X/Open Curses). |
| RETURN VALUES | The flushinp() function always returns OK. |
| ERRORS | None. |

NAME | fmod – floating-point remainder value function

SYNOPSIS | cc [*flag ...*] *file ...* -lm [*library ...*]
 | #include <math.h>
 |
 | double fmod(double x, double y);

DESCRIPTION | The **fmod()** function returns the floating-point remainder of the division of *x* by *y*.

RETURN VALUES | The **fmod()** function returns the value $x - i * y$, for some integer *i* such that, if *y* is non-zero, the result has the same sign as *x* and magnitude less than the magnitude of *y*.
 |
 | If *x* or *y* is NaN, NaN is returned.
 | If *y* is 0, NaN is returned and `errno` is set to EDOM.
 | If *x* is $\pm\text{Inf}$, NaN is returned.
 | If *y* is non-zero, `fmod(± 0 , y)` returns the value of *x*. If *x* is not $\pm\text{Inf}$, `fmod(x, $\pm\text{Inf}$)` returns the value of *x*.

ERRORS | The **fmod()** function may fail if:
 | **EDOM** *y* is 0.
 | No other errors will occur.

USAGE | Portable applications should not call **fmod()** with *y* equal to 0, because the result is implementation-dependent. The application should verify *y* is non-zero before calling **fmod()**.
 |
 | An application wishing to check for error situations should set `errno` to 0 before calling **fmod()**. If `errno` is non-zero on return, or the return value is NaN, an error has occurred.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **isnan(3M)**, **attributes(5)**

| | |
|--------------------|--|
| NAME | fmtmsg – display a message on stderr or system console |
| SYNOPSIS | <pre>#include <fmtmsg.h> int fmtmsg(long <i>classification</i>, const char *<i>label</i>, int <i>severity</i>, const char *<i>text</i>, const char *<i>action</i>, const char *<i>tag</i>);</pre> |
| DESCRIPTION | <p>The fmtmsg() function writes a formatted message to <code>stderr</code>, to the console, or to both, on a message's classification component. It can be used instead of the traditional <code>printf(3S)</code> interface to display messages to <code>stderr</code>, and in conjunction with <code>gettext(3C)</code>, provides a simple interface for producing language-independent applications.</p> <p>A formatted message consists of up to five standard components (<i>label</i>, <i>severity</i>, <i>text</i>, <i>action</i>, and <i>tag</i>) as described below. The <i>classification</i> component is not part of the standard message displayed to the user, but rather defines the source of the message and directs the display of the formatted message.</p> <p>classification Contains identifiers from the following groups of major classifications and subclassifications. Any one identifier from a subclass may be used in combination by ORing the values together with a single identifier from a different subclass. Two or more identifiers from the same subclass should not be used together, with the exception of identifiers from the display subclass. (Both display subclass identifiers may be used so that messages can be displayed to both <code>stderr</code> and the system console).</p> <ul style="list-style-type: none"> ■ “Major classifications” identify the source of the condition. Identifiers are: <code>MM_HARD</code> (hardware), <code>MM_SOFT</code> (software), and <code>MM_FIRM</code> (firmware). ■ “Message source subclassifications” identify the type of software in which the problem is spotted. Identifiers are: <code>MM_APPL</code> (application), <code>MM_UTIL</code> (utility), and <code>MM_OPSYS</code> (operating system). ■ “Display subclassifications” indicate where the message is to be displayed. Identifiers are: <code>MM_PRINT</code> to display the message on the standard error stream, <code>MM_CONSOLE</code> to display the message on the system console. Neither, either, or both identifiers may be used. ■ “Status subclassifications” indicate whether the application will recover from the condition. Identifiers are: <code>MM_RECOVER</code> (recoverable) and <code>MM_NRECOV</code> (non-recoverable). ■ An additional identifier, <code>MM_NULLMC</code>, indicates that no classification component is supplied for the message. |

| | |
|------------------------|---|
| <i>label</i> | Identifies the source of the message. The format of this component is two fields separated by a colon. The first field is up to 10 characters long; the second is up to 14 characters. Suggested usage is that <i>label</i> identifies the package in which the application resides as well as the program or application name. For example, the <i>label</i> <code>UX:cat</code> indicates the UNIX System V package and the <code>cat(1)</code> utility. |
| <i>severity</i> | <p>Indicates the seriousness of the condition. Identifiers for the standard levels of <i>severity</i> are:</p> <ul style="list-style-type: none"> ■ <code>MM_HALT</code> indicates that the application has encountered a severe fault and is halting. Produces the print string <code>HALT</code>. ■ <code>MM_ERROR</code> indicates that the application has detected a fault. Produces the print string <code>ERROR</code>. ■ <code>MM_WARNING</code> indicates a condition out of the ordinary that might be a problem and should be watched. Produces the print string <code>WARNING</code>. ■ <code>MM_INFO</code> provides information about a condition that is not in error. Produces the print string <code>INFO</code>. ■ <code>MM_NOSEV</code> indicates that no severity level is supplied for the message. <p>Other severity levels may be added by using the <code>addseverity()</code> routine.</p> |
| <i>text</i> | Describes the condition that produced the message. The <i>text</i> string is not limited to a specific size. |
| <i>action</i> | Describes the first step to be taken in the error recovery process. <code>fmtmsg()</code> precedes each action string with the prefix: <code>TOFIX: .</code> The <i>action</i> string is not limited to a specific size. |
| <i>tag</i> | An identifier which references on-line documentation for the message. Suggested usage is that <i>tag</i> includes the <i>label</i> and a unique identifying number. A sample <i>tag</i> is <code>UX:cat:146</code> . |

**Environment
Variables**

The `MSGVERB` and `SEV_LEVEL` environment variables control the behavior of **`fmtmsg()`** as follows:

MSGVERB

This variable determines which message components **fmtmsg()** selects when writing messages to `stderr`. Its value is a colon-separated list of optional keywords and can be set as follows:

```
MSGVERB=[keyword[:keyword[:...]]]  
export MSGVERB
```

Valid *keywords* are: `label`, `severity`, `text`, `action`, and `tag`. If `MSGVERB` contains a keyword for a component and the component's value is not the component's null value, **fmtmsg()** includes that component in the message when writing the message to `stderr`. If `MSGVERB` does not include a keyword for a message component, that component is not included in the display of the message. The keywords may appear in any order. If `MSGVERB` is not defined, if its value is the null string, if its value is not of the correct format, or if it contains keywords other than the valid ones listed above, **fmtmsg()** selects all components.

The first time **fmtmsg()** is called, it examines `MSGVERB` to determine which message components are to be selected when generating a message to write to the standard error stream, `stderr`. The values accepted on the initial call are saved for future calls.

The `MSGVERB` environment variable affects only those components that are selected for display to the standard error stream. All message components are included in console messages.

SEV_LEVEL

This variable defines severity levels and associates print strings with them for use by **fmtmsg()**. The standard severity levels listed below cannot be modified. Additional severity levels can also be defined, redefined, and removed using **addseverity()** (see **addseverity(3C)**). If the same severity level is defined by both **SEV_LEVEL** and **addseverity()**, the definition by **addseverity()** takes precedence.

- 0** (no severity is used)
- 1** HALT
- 2** ERROR
- 3** WARNING
- 4** INFO

The **SEV_LEVEL** variable can be set as follows:

```
SEV_LEVEL=[description[:description[:...]]]
export SEV_LEVEL
```

where *description* is a comma-separated list containing three fields:

description=severity_keyword,level,printstring

The *severity_keyword* field is a character string that is used as the keyword on the `-s severity` option to the **fmtmsg(1)** utility. (This field is not used by the **fmtmsg()** function.)

The *level* field is a character string that evaluates to a positive integer (other than 0, 1, 2, 3, or 4, which are reserved for the standard severity levels). If the keyword *severity_keyword* is used, *level* is the severity value passed on to the **fmtmsg()** function.

The *printstring* field is the character string used by **fmtmsg()** in the standard message format whenever the severity value *level* is used.

If a *description* in the colon list is not a three-field comma list, or if the second field of a comma list does not evaluate to a positive integer, that *description* in the colon list is ignored.

The first time **fmtmsg()** is called, it examines the SEV_LEVEL environment variable, if defined, to determine whether the environment expands the levels of severity beyond the five standard levels and those defined using **addseverity()**. The values accepted on the initial call are saved for future calls.

Use in Applications

One or more message components may be systematically omitted from messages generated by an application by using the null value of the argument for that component.

The table below indicates the null values and identifiers for **fmtmsg()** arguments.

| Argument | Type | Null-Value | Identifier |
|-----------------|-------|--------------|------------|
| <i>label</i> | char* | (char*) NULL | MM_NULLLBL |
| <i>severity</i> | int | 0 | MM_NULLSEV |
| <i>class</i> | long | 0L | MM_NULLMC |
| <i>text</i> | char* | (char*) NULL | MM_NULLTXT |
| <i>action</i> | char* | (char*) NULL | MM_NULLACT |
| <i>tag</i> | char* | (char*) NULL | MM_NULLTAG |

Another means of systematically omitting a component is by omitting the component keyword(s) when defining the MSGVERB environment variable (see the Environment Variables section above).

RETURN VALUES

The **fmtmsg()** returns the following values:

| | |
|----------|--|
| MM_OK | The function succeeded. |
| MM_NOTOK | The function failed completely. |
| MM_NOMSG | The function was unable to generate a message on the standard error stream, but otherwise succeeded. |
| MM_NOCON | The function was unable to generate a console message, but otherwise succeeded. |

EXAMPLES

EXAMPLE 1 The following example of **fmtmsg()**:

```
fmtmsg(MM_PRINT, "UX:cat", MM_ERROR, "invalid syntax",
"refer to manual", "UX:cat:001")
```

produces a complete message in the standard message format:

```
UX:cat: ERROR: invalid syntax
TO FIX: refer to manual   UX:cat:001
```

EXAMPLE 2 When the environment variable MSGVERB is set as follows:

```
MSGVERB=severity:text:action
```

and the Example 1 is used, **fmtmsg()** produces:

```
ERROR: invalid syntax
TO FIX: refer to manual
```

EXAMPLE 3 When the environment variable SEV_LEVEL is set as follows:

```
SEV_LEVEL=note,5,NOTE
```

the following call to **fmtmsg()**

```
fmtmsg(MM_UTIL | MM_PRINT, "UX:cat", 5, "invalid syntax",
"refer to manual", "UX:cat:001")
```

produces

```
UX:cat: NOTE: invalid syntax
TO FIX: refer to manual   UX:cat:001
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO `fmtmsg(1)`, `addseverity(3C)`, `gettext(3C)`, `printf(3S)`, `attributes(5)`

| NAME | fn_attr_bind – bind a reference to a name and associate attributes with named object | | | | |
|----------------------|---|------------------|--------------------------------------|----------|---------|
| SYNOPSIS | <pre>#include <xfn/xfn.h> int fn_attr_bind(FN_ctx_t *ctx, const FN_composite_name_t *name, const FN_ref_t *ref, const FN_attrset_t *attrs, unsigned int exclusive, FN_status_t *status);</pre> | | | | |
| DESCRIPTION | <p>This operation binds the supplied reference <i>ref</i> to the supplied composite name <i>name</i> relative to <i>ctx</i>, and associates the attributes specified in <i>attrs</i> with the named object. The binding is made in the target context, that is, that context named by all but the terminal atomic part of <i>name</i>. The operation binds the terminal atomic name to the supplied reference in the target context. The target context must already exist.</p> <p>The value of <i>exclusive</i> determines what happens if the terminal atomic part of the name is already bound in the target context. If <i>exclusive</i> is nonzero and <i>name</i> is already bound, the operation fails. If <i>exclusive</i> is 0, the new binding replaces any existing binding, and, if <i>attrs</i> is not NULL, <i>attrs</i> replaces any existing attributes associated with the named object. If <i>attrs</i> is NULL and <i>exclusive</i> is 0, any existing attributes associated with the named object are left unchanged.</p> | | | | |
| RETURN VALUES | fn_attr_bind() returns 1 upon success, 0 upon failure. | | | | |
| ERRORS | <p>fn_attr_bind() sets <i>status</i> as described in FN_status_t(3N) and xfn_status_codes(3N). Of special relevance for this operation is the following status code:</p> <table border="0"> <tr> <td style="padding-right: 20px;">FN_E_NAME_IN_USE</td> <td>The supplied name is already in use.</td> </tr> </table> | FN_E_NAME_IN_USE | The supplied name is already in use. | | |
| FN_E_NAME_IN_USE | The supplied name is already in use. | | | | |
| USAGE | <p>The value of <i>ref</i> cannot be NULL. If the intent is to reserve a name using fn_attr_bind(), a reference containing no address should be supplied. This reference may be name service-specific or it may be the conventional NULL reference.</p> <p>If multiple sources are updating a reference or attributes associated with a named object, they must synchronize amongst each other when adding, modifying, or removing from the address list of a bound reference, or manipulating attributes associated with the named object.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |

SEE ALSO

FN_composite_name_t(3N), FN_ctx_t(3N), FN_ref_t(3N),
FN_status_t(3N), fn_ctx_bind(3N), fn_ctx_lookup(3N),
fn_ctx_unbind(3N), xfn_attributes(3N), xfn_status_codes(3N),
attributes(5)

| NAME | fn_attr_create_subcontext – create a subcontext in a context and associate attributes with newly created context | | | | |
|----------------------|--|------------------|--|----------|---------|
| SYNOPSIS | <pre>#include <xfn/xfn.h> FN_ref_t *fn_attr_create_subcontext(FN_ctx_t *ctx, const FN_composite_name_t *name, const FN_attrset_t *attrs, FN_status_t *status);</pre> | | | | |
| DESCRIPTION | <p>This operation creates a new XFN context of the same type as the target context, that is, that context named by all but the terminal atomic component of <i>name</i>, and binds it to the supplied composite name. In addition, attributes given in <i>attrs</i> are associated with the newly created context.</p> <p>The target context must already exist. The new context is created and bound in the target context using the terminal atomic name in <i>name</i>. The operation returns a reference to the newly created context.</p> | | | | |
| RETURN VALUES | fn_attr_create_subcontext() returns a reference to the newly created context; if the operation fails, it returns a NULL pointer. | | | | |
| ERRORS | <p>fn_attr_create_subcontext() sets <i>status</i> as described in FN_status_t(3N) and xfn_status_codes(3N). Of special relevance for this operation is the following status code:</p> <table border="0"> <tr> <td style="padding-right: 20px;">FN_E_NAME_IN_USE</td> <td>The terminal atomic name already exists in the target context.</td> </tr> </table> | FN_E_NAME_IN_USE | The terminal atomic name already exists in the target context. | | |
| FN_E_NAME_IN_USE | The terminal atomic name already exists in the target context. | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | <p>FN_composite_name_t(3N), FN_ctx_t(3N), FN_ref_t(3N), FN_status_t(3N), fn_attr_bind(3N), fn_ctx_bind(3N), fn_ctx_create_subcontext(3N), fn_ctx_destroy_subcontext(3N), fn_ctx_lookup(3N), xfn_attributes(3N), xfn_status_codes(3N), attributes(5)</p> | | | | |

| | |
|--------------------|--|
| NAME | fn_attr_ext_search, FN_ext_searchlist_t, fn_ext_searchlist_next, fn_ext_searchlist_destroy – search for names in the specified context(s) whose attributes satisfy the filter |
| SYNOPSIS | <pre>#include <xfn/xfn.h> FN_ext_searchlist_t * fn_attr_ext_search(FN_ctx_t * ctx, const FN_composite_name_t * name, const FN_search_control_t * control, const FN_search_filter_t * filter, FN_status_t * status); FN_composite_name_t * fn_ext_searchlist_next(FN_ext_searchlist_t * esl, FN_ref_t ** returned_ref, FN_attrset_t ** returned_attrs, FN_status_t * status); void fn_ext_searchlist_destroy(FN_ext_searchlist_t * esl);</pre> |
| DESCRIPTION | <p>This set of operations is used to list names of objects whose attributes satisfy the filter expression. The references to which these names are bound and specified attributes and their values may also be returned.</p> <p><i>control</i> encapsulates the option settings for the search. These options are:</p> <ul style="list-style-type: none"> ■ the scope of the search ■ whether XFN links are followed ■ a limit on the number of names returned ■ whether references and specific attributes associated with the named objects that satisfy the filter are returned <p>The scope of the search is one of:</p> <ul style="list-style-type: none"> ■ the object named <i>name</i> relative to the context <i>ctx</i> ■ the context named <i>name</i> relative to the context <i>ctx</i> ■ the context named <i>name</i> relative to the context <i>ctx</i> , and its subcontexts <p>or</p> <ul style="list-style-type: none"> ■ the context named <i>name</i> relative to the context <i>ctx</i> , and a context implementation-defined set of subcontexts <p>If the value of <i>control</i> is 0 , default control option settings are used. The default settings are:</p> <ul style="list-style-type: none"> ■ scope is search named context ■ links are not followed |

- all names of objects that satisfy the filter are returned
- references and attributes are not returned

The `FN_search_control_t` type is described in `FN_search_control_t(3N)`.

The filter expression *filter* in `fn_attr_ext_search()` is evaluated against the attributes of the objects bound in the scope of the search. The filter evaluates to either `TRUE` or `FALSE`. The names and, optionally, the references and attributes of objects whose attributes satisfy the filter are enumerated. If the value of *filter* is 0, all names within the search scope are enumerated. The `FN_search_filter_t` type is described in `FN_search_filter_t(3N)`.

The call to `fn_attr_ext_search()` initiates the search process. It returns a handle to an `FN_ext_searchlist_t` object that is used to enumerate the names of the objects that satisfy the filter.

The operation `fn_ext_searchlist_next()` returns the next name in the enumeration identified by *esl*; it also updates *esl* to indicate the state of the enumeration. If the reference to which the name is bound was requested, it is returned in *returned_ref*. Requested attributes associated with the name are returned in *returned_attrs*; each attribute consists of an attribute identifier, syntax, and value(s). Successive calls to `fn_ext_searchlist_next()` using *esl* return successive names and, optionally, their references and attributes, in the enumeration; these calls further update the state of the enumeration.

The names that are returned are composite names, to be resolved relative to the starting context for the search. This starting context is the context named *name* relative to *ctx* unless the scope of the search is only the named object. If the scope of the search is only the named object, the terminal atomic name in *name* is returned.

`fn_ext_searchlist_destroy()` releases resources used during the enumeration. This may be invoked at any time to terminate the enumeration.

RETURN VALUES

`fn_attr_ext_search()` returns a pointer to an `FN_ext_searchlist_t` object if the search is successfully initiated; it returns a `NULL` pointer if the search cannot be initiated or if no named object with attributes whose values satisfy the filter expression is found.

`fn_ext_searchlist_next()` returns a pointer to an `FN_composite_name_t` object (see `FN_composite_name_t(3N)`) that is the next name in the enumeration; it returns a `NULL` pointer if no more names can be returned. If *returned_attrs* is a `NULL` pointer, no attributes are returned; otherwise, *returned_attrs* contains the attributes associated with the named object, as specified in the control parameter to `fn_attr_ext_search()`. If *returned_ref* is a

ERRORS

NULL pointer, no reference is returned; otherwise, if *control* specified the return of the reference of the named object, that reference is returned in *returned_ref* .

In the case of a failure, these operations return in the *status* argument a code indicating the nature of the failure.

If successful, **fn_attr_ext_search()** returns a pointer to an `FN_ext_searchlist_t` object and sets *status* to `FN_SUCCESS` .

fn_attr_ext_search() returns a NULL pointer when no more names can be returned. *status* is set in the following way:

| | |
|---|---|
| <code>FN_SUCCESS</code> | A named object could not be found whose attributes satisfied the filter expression. |
| <code>FN_E_NOT_A_CONTEXT</code> | The object named for the start of the search was not a context and the search scope was the given context or the given context and its subcontexts. |
| <code>FN_E_SEARCH_INVALID_FILTER</code> | The filter could not be evaluated TRUE or FALSE , or there was some other problem with the filter. |
| <code>FN_E_SEARCH_INVALID_OPTION</code> | A supplied search control option could not be supported. |
| <code>FN_E_SEARCH_INVALID_OP</code> | An operator in the filter expression is not supported or, if the operator is an extended operator, the number of types of arguments supplied does not match the signature of the operation. |
| <code>FN_E_ATTR_NO_PERMISSION</code> | The caller did not have permission to read one or more of the attributes specified in the filter. |
| <code>FN_E_INVALID_ATTR_VALUE</code> | A value type in the filter did not match the syntax of the attribute against which it was being evaluated. |

Other status codes are possible as described in `FN_status_t(3N)` and `xfn_status_codes(3N)` .

Each successful call to **fn_ext_searchlist_next()** returns a name and, optionally, its reference in *returned_ref* and requested attributes in *returned_attrs* . *status* is set in the following way:

| | |
|--|--|
| FN_SUCCESS | All requested attributes were returned successfully with the name. |
| FN_E_ATTR_NO_PERMISSION | The caller did not have permission to read one or more of the requested attributes. |
| FN_E_INVALID_ATTR_IDENTIFIER | A requested attribute identifier was not in a format acceptable to the naming system, or its contents were not valid for the format specified. |
| FN_E_NO_SUCH_ATTRIBUTE | The named object did not have one of the requested attributes. |
| FN_E_INSUFFICIENT_RESOURCES | Insufficient resources are available to return all the requested attributes and their values. |
| FN_E_ATTR_NO_PERMISSION | These indicate that some of the requested attributes may have been returned in <i>returned_attrs</i> but one or more of them could not be returned. Use <code>fn_attr_get(3N)</code> or <code>fn_attr_multi_get(3N)</code> to discover why these attributes could not be returned. |
| FN_E_INVALID_ATTR_IDENTIFIER | |
| FN_E_NO_SUCH_ATTRIBUTE | |
| FN_E_INSUFFICIENT_RESOURCES | |
| | |
| If <code>fn_ext_searchlist_next()</code> returns a name, it can be called again to get the next name in the enumeration. | |
| <code>fn_ext_searchlist_next()</code> returns a <code>NULL</code> pointer if no more names can be returned. <i>status</i> is set in the following way: | |
| FN_SUCCESS | The search has completed successfully. |
| FN_E_PARTIAL_RESULT | The enumeration is not yet complete but cannot be continued. |
| FN_E_ATTR_NO_PERMISSION | The caller did not have permission to read one or more of the attributes specified in the filter. |

FN_E_INVALID_ENUM_HANDLE

The supplied enumeration handle was not valid. Possible reasons could be that the handle was from another enumeration, or the context being enumerated no longer accepts the handle (due to such events as handle expiration or updates to the context).

Other status codes are possible as described in `FN_status_t(3N)` and `xfn_status_codes(3N)`.

USAGE

The search performed by `fn_attr_ext_search()` is not ordered in any way, including the traversal of subcontexts. The names enumerated using `fn_ext_searchlist_next()` are not ordered in any way. Furthermore, there is no guarantee that any two series of enumerations with the same arguments to `fn_attr_ext_search()` will return the names in the same order.

XFN links encountered during the resolution of *name* are followed, regardless of the follow links control setting, and the search starts at the final named object or context.

If *control* specifies that the search should follow links, XFN link names encountered during the search are followed and the terminal named object is searched. If the terminal named object is bound to a context and the scope of the search includes subcontexts, that context and its subcontexts are also searched. For example, if *aname* is bound to an XFN link, *lname*, in a context within the scope of the search, and *aname* is returned by `fn_ext_searchlist_next()`, this means that the object identified by *lname* satisfied the filter expression. *aname* is returned instead of *lname* because *aname* can always be named relative to the starting context for the search.

If *control* specifies that the search should not follow links, the attributes associated with the names of XFN links are searched. For example, if *aname* is bound to an XFN link, *lname*, in a context within the scope of the search, and *aname* is returned by `fn_ext_searchlist_next()`, this means that the object identified by *aname* satisfied the filter expression.

When following XFN links, `fn_attr_ext_search()` may search contexts outside of *scope*. In addition, if the link name's terminal atomic name is bound in a context within *scope*, the operation may return the same object more than once.

XFN does not specify how *control* affects the following of native naming system links during the search.

EXAMPLES

EXAMPLE 1 A sample program of displaying how the `fn_attr_ext_search()` operation may be used.

The following code fragment illustrates how the `fn_attr_ext_search()` operation may be used. The code consists of three parts: preparing the arguments for the search, performing the search, and cleaning up.

The first part involves getting the name of the context to start the search and constructing the search filter that named objects in the context must satisfy. This is done in the declarations part of the code and by the routine `get_search_query`. See `FN_search_filter_t(3N)` for the description of *sfilter* and the filter creation operation.

The next part involves doing the search and enumerating the results of the search. This is done by first getting a context handle to the Initial Context, and then passing that handle along with the name of the target context and search filter to `fn_attr_ext_search()`. This particular call to `fn_attr_ext_search()` uses the default search control options (by passing in 0 as the *control* argument). This means that the search will be performed in the context named by *target_name* and that no reference or attributes will be returned. In addition, any XFN links encountered will not be followed and all named objects that satisfy the search filter will be returned (that is, no limit). If successful, `fn_attr_ext_search()` returns *esl*, a handle for enumerating the results of the search. The results of the search are enumerated using calls to `fn_ext_searchlist_next()`, which returns the name of the object. (The arguments *returned_ref* and *returned_attrs* to `fn_ext_searchlist_next()` are 0 because the default search control used in `fn_attr_ext_search()` did not request them to be returned.)

The last part of the code involves cleaning up the resources used during the search and enumeration. The call to `fn_ext_searchlist_destroy()` releases resources reserved for this enumeration. The other calls release the context handle, name, filter, and status objects created earlier.

```

/* Declarations */
FN_ctx_t *ctx;
FN_ext_searchlist_t *esl;
FN_composite_name_t *name;
FN_status_t *status = fn_status_create();
FN_composite_name_t *target_name = get_name_from_user_input();
FN_search_filter_t *sfilter = get_search_query();
/* Get context handle to Initial Context */
ctx = fn_ctx_handle_from_initial(status);
/* error checking on 'status' */
/* Initiate search */
if ((esl=fn_attr_ext_search(ctx, target_name,
\011/* default controls */ 0, sfilter, status)) == 0) {
\011/* report 'status', cleanup, and exit */
}
/* Enumerate names requested */
while (name=fn_ext_searchlist_next(esl, 0, 0, status)) {

```

```

\011/* do something with 'name' */
\011fn_composite_destroy(name);
}
/* check 'status' for reason for end of enumeration */
/* Clean up */
fn_ext_searchlist_destroy(esl);
fn_search_filter_destroy(sfilter);
fn_ctx_handle_destroy(ctx);
fn_composite_name_destroy(target_name);
fn_status_destroy(status);
/*
* Procedure for constructing the filter object for search:
* \011"age" attribute is greater than or equal to 17 AND
*\011\011less than or equal to 25
*\011AND the "student" attribute is present.
*/
FN_search_filter_t *
get_search_query()
{
\011extern FN_attribute_t *attr_age;
\011extern FN_attribute_t *attr_student;
\011FN_search_filter_t *sfilter;
\011unsigned int filter_status;
\011sfilter = fn_search_filter_create(
\011\011&filter_status,
\011\011"(%a >= 17) and (%a <= 25) and %a",
\011\011attr_age, attr_age, attr_student);
\011/* error checking on 'filter_status' */
\011return (sfilter);
}

```

ATTRIBUTES

See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

[FN_attrset_t\(3N\)](#), [FN_composite_name_t\(3N\)](#), [FN_ctx_t\(3N\)](#),
[FN_ref_t\(3N\)](#), [FN_search_control_t\(3N\)](#), [FN_search_filter_t\(3N\)](#),
[FN_status_t\(3N\)](#), [fn_attr_get\(3N\)](#), [fn_attr_multi_get\(3N\)](#),
[xfn_status_codes\(3N\)](#), [attributes\(5\)](#)

| NAME | fn_attr_get – return specified attribute associated with name | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lxfn [library ...] #include <xfn/xfn.h></pre> <p>FN_attribute_t *fn_attr_get(FN_ctx_t *ctx, const FN_composite_name_t *name, const FN_identifier_t *attribute_id, unsigned int follow_link, FN_status_t *status);</p> | | | | |
| DESCRIPTION | <p>This operation returns the identifier, syntax and values of a specified attribute for the object named <i>name</i> relative to <i>ctx</i>. If <i>name</i> is empty, the attribute associated with <i>ctx</i> is returned.</p> <p>The value of <i>follow_link</i> determines what happens when the terminal atomic part of <i>name</i> is bound to an XFN link. If <i>follow_link</i> is non-zero, such a link is followed, and the values of the attribute associated with the final named object are returned; if <i>follow_link</i> is zero, such a link is not followed. Any XFN links encountered before the terminal atomic name are always followed.</p> | | | | |
| RETURN VALUES | fn_attr_get returns a pointer to an FN_attribute_t object if the operation succeeds; it returns a NULL pointer (0) if the operation fails. | | | | |
| ERRORS | fn_attr_get() sets <i>status</i> as described in FN_status_t(3N) and xfn_status_codes(3N). | | | | |
| USAGE | fn_attr_get_values() and its related operations are used for getting individual values of an attribute. They should be used if the combined size of all the values are expected to be too large to be returned in a single invocation of fn_attr_get(). | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | FN_attribute_t(3N), FN_composite_name_t(3N), FN_ctx_t(3N), FN_identifier_t(3N), FN_status_t(3N), fn_attr_get_values(3N), xfn(3N), xfn_attributes(3N), xfn_status_codes(3N), attributes(5) | | | | |
| NOTES | The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward | | | | |

standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

| NAME | fn_attr_get_ids – get a list of the identifiers of all attributes associated with named object | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lxfn [library ...] #include <xfn/xfn.h></pre> <p>FN_attrset_t *fn_attr_get_ids(FN_ctx_t *ctx, const FN_composite_name_t *name, unsigned int follow_link, FN_status_t *status);</p> | | | | |
| DESCRIPTION | <p>This operation returns a list of the attribute identifiers of all attributes associated with the object named by <i>name</i> relative to the context <i>ctx</i>. If <i>name</i> is empty, the attribute identifiers associated with <i>ctx</i> are returned.</p> <p>The value of <i>follow_link</i> determines what happens when the terminal atomic part of <i>name</i> is bound to an XFN link. If <i>follow_link</i> is non-zero, such a link is followed, and the values of the attribute associated with the final named object are returned; if <i>follow_link</i> is zero, such a link is not followed. Any XFN links encountered before the terminal atomic name are always followed.</p> | | | | |
| RETURN VALUES | This operation returns a pointer to an object of type FN_attrset_t; if the operation fails, a NULL pointer (0) is returned. | | | | |
| ERRORS | This operation sets <i>status</i> as described in FN_status_t(3N) and xfn_status_codes(3N). | | | | |
| USAGE | The attributes in the returned set do not contain the syntax or values of the attributes, only their identifiers. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | FN_attribute_t(3N), FN_attrset_t(3N), FN_composite_name_t(3N), FN_ctx_t(3N), FN_status_t(3N), fn_attr_get(3N), fn_attr_multi_get(3N) xfn(3N), xfn_attributes(3N), xfn_status_codes(3N), attributes(5) | | | | |
| NOTES | The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward | | | | |

standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

NAME | fn_attr_get_values, FN_valuelist_t, fn_valuelist_next, fn_valuelist_destroy –
return values of an attribute

SYNOPSIS

```
cc
[
  flag
  ... ]
file
...
-lxfn
[
  library
  ... ]
#include <xfn/xfn.h>

FN_valuelist_t * fn_attr_get_values(FN_ctx_t * ctx, const FN_composite_name_t *
name, const FN_identifier_t * attribute_id, unsigned int follow_link, FN_status_t * status);

FN_attrvalue_t * fn_valuelist_next(FN_valuelist_t * vl, FN_identifier_t **
attr_syntax, FN_status_t * status);

void fn_valuelist_destroy(FN_valuelist_t * vl, FN_status_t * status);
```

DESCRIPTION

This set of operations is used to obtain the values of a single attribute, identified by *attribute_id*, associated with the object named *name*, resolved in the context *ctx*. If *name* is empty, the attribute values associated with *ctx* are obtained.

The value of *follow_link* determines what happens when the terminal atomic part of *name* is bound to an XFN link. If *follow_link* is non-zero, such a link is followed, and the values of the attribute associated with the final named object are returned; if *follow_link* is zero, such a link is not followed. Any XFN links encountered before the terminal atomic name are always followed.

The operation **fn_attr_get_values()** initiates the enumeration process. It returns a handle to an `FN_valuelist_t` object that can be used to enumerate the values of the specified attribute.

The operation **fn_valuelist_next()** returns a new `FN_attrvalue_t` object containing the next value in the attribute and may be called multiple times until all values are retrieved. The syntax of the attribute is returned in *attr_syntax*.

The operation **fn_valuelist_destroy()** is used to release the resources used during the enumeration. This may be invoked before the enumeration has completed to terminate the enumeration.

RETURN VALUES

These operations work in a fashion similar to the **fn_ctx_list_names()** operations.

fn_attr_get_values() returns a pointer to an `FN_valuelist_t` object if the enumeration process is successfully initiated; it returns a `NULL` pointer if the process failed.

fn_valuelist_next() returns a `NULL` pointer if no more attribute values can be returned.

In the case of a failure, these operations set *status* to indicate the nature of the failure.

ERRORS

Each successful call to **fn_valuelist_next()** returns an attribute value. *status* is set to `FN_SUCCESS` .

When **fn_valuelist_next()** returns a `NULL` pointer, it indicates that no more values can be returned. *status* is set in the following way:

| | |
|---------------------------------------|--|
| <code>FN_SUCCESS</code> | The enumeration has completed successfully. |
| <code>FN_E_INVALID_ENUM_HANDLE</code> | The given enumeration handle is not valid. Possible reasons could be that the handle was from another enumeration, or the context being enumerated no longer accepts the handle (due to such events as handle expiration or updates to the context). |
| <code>FN_E_PARTIAL_RESULT</code> | The enumeration is not yet complete but cannot be continued. |

In addition to these status codes, other status codes are also possible in calls to these operations. In such cases, *status* is set as described in **FN_status_t(3N)** and **xfn_status_codes(3N)** .

USAGE

This interface should be used instead of **fn_attr_get()** if the combined size of all the values is expected to be too large to be returned by **fn_attr_get()** .

There may be a relationship between the *ctx* argument supplied to **fn_attr_get_values()** and the `FN_valuelist_t` object it returns. For example, some implementations may store the context handle *ctx* within the `FN_valuelist_t` object for subsequent **fn_valuelist_next()** calls. In general, an **fn_ctx_handle_destroy(3N)** should not be invoked on *ctx* until the enumeration has terminated.

ATTRIBUTES

See **attributes** (5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

FN_attribute_t(3N), **FN_attrvalue_t**(3N),
FN_composite_name_t(3N), **FN_ctx_t**(3N), **FN_identifier_t**(3N),
FN_status_t(3N), **fn_attr_get**(3N), **fn_ctx_handle_destroy**(3N),
fn_ctx_list_names(3N), **xfn**(3N), **xfn_attributes**(3N),
xfn_status_codes(3N), **attributes**(5)

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

NAME FN_attribute_t, fn_attribute_create, fn_attribute_destroy, fn_attribute_copy, fn_attribute_assign, fn_attribute_identifier, fn_attribute_syntax, fn_attribute_valuecount, fn_attribute_first, fn_attribute_next, fn_attribute_add, fn_attribute_remove – an XFN attribute

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lxfn
[
library
... ]
#include <xfn/xfn.h>

FN_attribute_t * fn_attribute_create(const FN_identifier_t * attribute_id, const
FN_identifier_t * attribute_syntax);

void fn_attribute_destroy(FN_attribute_t * attr);

FN_attribute_t * fn_attribute_copy(const FN_attribute_t * attr);

FN_attribute_t * fn_attribute_assign(FN_attribute_t * dst, const FN_attribute_t *
src);

const FN_identifier_t * fn_attribute_identifier(const FN_attribute_t * attr);

const FN_identifier_t * fn_attribute_syntax(const FN_attribute_t * attr);

unsigned int fn_attribute_valuecount(const FN_attribute_t * attr);

const FN_attrvalue_t * fn_attribute_first(const FN_attribute_t * attr, void **
iter_pos);

const FN_attrvalue_t * fn_attribute_next(const FN_attribute_t * attr, void ** iter_pos);

int fn_attribute_add(FN_attribute_t * attr, const FN_attrvalue_t * attribute_value,
unsigned int exclusive);

int fn_attribute_remove(FN_attribute_t * attr, const FN_attrvalue_t * attribute_value);
```

DESCRIPTION

An attribute has an attribute identifier, a syntax, and a set of distinct values. Each value is a sequence of octets. The operations associated with objects of type FN_attribute_t allow the construction, destruction, and manipulation of an attribute and its value set.

The attribute identifier and its syntax are specified using an `FN_identifier_t`. **fn_attribute_create()** creates a new attribute object with the given identifier and syntax, and an empty set of values. **fn_attribute_destroy()** releases the storage associated with *attr*. **fn_attribute_copy()** returns a copy of the object pointed to by *attr*. **fn_attribute_assign()** makes a copy of the attribute object pointed to by *src* and assigns it to *dst*, releasing any old contents of *dst*. A pointer to the same object as *dst* is returned.

fn_attribute_identifier() returns the attribute identifier of *attr*.
fn_attribute_syntax() returns the attribute syntax of *attr*.
fn_attribute_valuecount() returns the number of attribute values in *attr*.

fn_attribute_first() and **fn_attribute_next()** are used to enumerate the values of an attribute. Enumeration of the values of an attribute may return the values in any order. **fn_attribute_first()** returns an attribute value from *attr* and sets the iteration marker *iter_pos*. Subsequent calls to **fn_attribute_next()** returns the next attribute value identified by *iter_pos* and advances *iter_pos*. Adding or removing values from an attribute invalidates any iteration markers that the caller holds.

fn_attribute_add() adds a new value *attribute_value* to *attr*. The operation succeeds (but no change is made) if *attribute_value* is already in *attr* and *exclusive* is 0; the operation fails if *attribute_value* is already in *attr* and *exclusive* is non-zero.

fn_attribute_remove() removes *attribute_value* from *attr*. The operation succeeds even if *attribute_value* is not amongst *attr*'s values.

RETURN VALUES

fn_attribute_first() returns 0 if the attribute contains no values.
fn_attribute_next() returns 0 if there are no more values to be returned in the attribute (as identified by the iteration marker) or if the iteration marker is invalid.

fn_attribute_add() and **fn_attribute_remove()** return 1 if the operation succeeds, 0 if it fails.

USAGE

Manipulation of attributes using the operations described in this manual page does not affect their representation in the underlying naming system. Changes to attributes in the underlying naming system can only be effected through the use of the interfaces described in `xfn_attributes(3N)`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO `FN_attrset_t(3N)`, `FN_attrvalue_t(3N)`, `FN_identifier_t(3N)`, `fn_attr_get(3N)`, `fn_attr_modify(3N)`, `xfn(3N)`, `xfn_attributes(3N)`, `attributes(5)`

NOTES The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

| | | | | | | | |
|--------------------------|--|----------------|---|--------------------------|--|-------------------|--|
| NAME | fn_attr_modify – modify specified attribute associated with name | | | | | | |
| SYNOPSIS | <pre>cc [flag ...] file ... -lxfn [library ...] #include <xfn/xfn.h></pre> <p>int fn_attr_modify(FN_ctx_t *ctx, const FN_composite_name_t *name, unsigned int mod_op, const FN_attribute_t *attr, unsigned int follow_link, FN_status_t *status);</p> | | | | | | |
| DESCRIPTION | <p>This operation modifies according to <i>mod_op</i> the attribute <i>attr</i> associated with the object named <i>name</i> relative to <i>ctx</i>. If <i>name</i> is empty, the attribute associated with <i>ctx</i> is modified.</p> <p>The value of <i>follow_link</i> determines what happens when the terminal atomic part of <i>name</i> is bound to an XFN link. If <i>follow_link</i> is non-zero, such a link is followed, and the values of the attribute associated with the final named object are returned; if <i>follow_link</i> is zero, such a link is not followed. Any XFN links encountered before the terminal atomic name are always followed.</p> <p>The modification is made on the attribute identified by the attribute identifier of <i>attr</i>. The syntax and values of <i>attr</i> are used according to the modification operation.</p> <p>The modification operations are as follows:</p> <table border="0" style="width: 100%;"> <tr> <td style="vertical-align: top; padding-right: 20px;">FN_ATTR_OP_ADD</td> <td>Add an attribute with given attribute identifier and set of values. If an attribute with this identifier already exists, replace the set of values with those in the given set. The set of values may be empty if the target naming system permits.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 20px;">FN_ATTR_OP_ADD_EXCLUSIVE</td> <td>Add an attribute with the given attribute identifier and set of values. The operation fails if an attribute with this identifier already exists. The set of values may be empty if the target naming system permits.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 20px;">FN_ATTR_OP_REMOVE</td> <td>Remove the attribute with the given attribute identifier and all of its values. The operation succeeds even if the attribute does not exist. The values of the attribute supplied with this operation are ignored.</td> </tr> </table> | FN_ATTR_OP_ADD | Add an attribute with given attribute identifier and set of values. If an attribute with this identifier already exists, replace the set of values with those in the given set. The set of values may be empty if the target naming system permits. | FN_ATTR_OP_ADD_EXCLUSIVE | Add an attribute with the given attribute identifier and set of values. The operation fails if an attribute with this identifier already exists. The set of values may be empty if the target naming system permits. | FN_ATTR_OP_REMOVE | Remove the attribute with the given attribute identifier and all of its values. The operation succeeds even if the attribute does not exist. The values of the attribute supplied with this operation are ignored. |
| FN_ATTR_OP_ADD | Add an attribute with given attribute identifier and set of values. If an attribute with this identifier already exists, replace the set of values with those in the given set. The set of values may be empty if the target naming system permits. | | | | | | |
| FN_ATTR_OP_ADD_EXCLUSIVE | Add an attribute with the given attribute identifier and set of values. The operation fails if an attribute with this identifier already exists. The set of values may be empty if the target naming system permits. | | | | | | |
| FN_ATTR_OP_REMOVE | Remove the attribute with the given attribute identifier and all of its values. The operation succeeds even if the attribute does not exist. The values of the attribute supplied with this operation are ignored. | | | | | | |

FN_ATTR_OP_ADD_VALUES

Add the given values to those of the given attribute (resulting in the attribute having the union of its prior value set with the set given). Create the attribute if it does not exist already. The set of values may be empty if the target naming system permits.

FN_ATTR_OP_REMOVE_VALUES

Remove the given values from those of the given attribute (resulting in the attribute having the set difference of its prior value set and the set given). This succeeds even if some of the given values are not in the set of values that the attribute has. In naming systems that require an attribute to have at least one value, removing the last value will remove the attribute as well.

RETURN VALUES

1 Successful operation.
0 Operation failed.

ERRORS

fn_attr_modify() sets *status* as described in **FN_status_t(3N)** and **xfn_status_codes(3N)**.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

FN_attribute_t(3N), **FN_composite_name_t(3N)**, **FN_ctx_t(3N)**, **FN_status_t(3N)**, **fn_attr_multi_modify(3N)**, **xfn(3N)**, **xfn_attributes(3N)**, **xfn_status_codes(3N)**, **attributes(5)**

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications

developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

| | |
|--------------------|--|
| NAME | FN_attrmodlist_t, fn_attrmodlist_create, fn_attrmodlist_destroy, fn_attrmodlist_copy, fn_attrmodlist_assign, fn_attrmodlist_count, fn_attrmodlist_first, fn_attrmodlist_next, fn_attrmodlist_add – a list of attribute modifications |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lxfn [<i>library</i> ...] #include <xfn/xfn.h> FN_attrmodlist_t * fn_attrmodlist_create(void); void fn_attrmodlist_destroy(FN_attrmodlist_t * <i>modlist</i>); FN_attrmodlist_t * fn_attrmodlist_copy(const FN_attrmodlist_t * <i>modlist</i>); FN_attrmodlist_t * fn_attrmodlist_assign(FN_attrmodlist_t * <i>dst</i>, const FN_attrmodlist_t * <i>src</i>); unsigned int fn_attrmodlist_count(const FN_attrmodlist_t * <i>modlist</i>); const FN_attribute_t * fn_attrmodlist_first(const FN_attrmodlist_t * <i>modlist</i>, void ** <i>iter_pos</i>, unsigned int * <i>first_mod_op</i>); const FN_attribute_t * fn_attrmodlist_next(const FN_attrmodlist_t * <i>modlist</i>, void ** <i>iter_pos</i>, unsigned int * <i>mod_op</i>); int fn_attrmodlist_add(FN_attrmodlist_t * <i>modlist</i>, unsigned int <i>mod_op</i>, const FN_attribute_t * <i>attr</i>); </pre> |
| DESCRIPTION | <p>An attribute modification list allows for multiple modification operations to be made on the attributes associated with a single named object. It is used in the <code>fn_attr_multi_modify(3N)</code> operation.</p> <p>An attribute modification list is a list of attribute modification specifiers. An attribute modification specifier consists of an attribute object and an operation specifier. The attribute's identifier indicates the attribute that is to be operated upon. The attribute's values are used in a manner depending on the operation. The operation specifier is an <code>unsigned int</code> that must have one of the values:</p> <pre> FN_ATTR_OP_ADD </pre> |

```
FN_ATTR_OP_ADD_EXCLUSIVE
```

```
FN_ATTR_OP_REMOVE
```

```
FN_ATTR_OP_ADD_VALUES
```

or

```
FN_ATTR_OP_REMOVE_VALUES
```

(See `fn_attr_modify(3N)` for detailed descriptions of these specifiers.) The operations are to be performed in the order in which they appear in the modification list.

`fn_attrmodlist_create()` creates an empty attribute modification list.

`fn_attrmodlist_destroy()` releases the storage associated with `modlist`.

`fn_attrmodlist_copy()` returns a copy of the attribute modification list `modlist`.

`fn_attrmodlist_assign()` makes a copy of `src` and assigns it to `dst`, releasing any old contents of `dst`. It returns a pointer to the same object as `dst`.

`fn_attrmodlist_count()` returns the number attribute modification items in the attribute modification list.

The iterators `fn_attrmodlist_first()` and `fn_attrmodlist_next()` return a handle to the attribute part of the modification and return the operation specifier part through an `unsigned int *` parameter. `fn_attrmodlist_first()` returns the attribute of the first modification item from `modlist` and sets `mod_op` to be the code of the modification operation of that item; `iter_pos` is set after the first modification item.

`fn_attrmodlist_next()` returns the attribute of the next modification item from `modlist` after `iter_pos` and advances `iter_pos`; `mod_op` is set to the code of the modification operation of that item. The order of the items returned during an enumeration is the same as the order by which the items were added to the modification list.

`fn_attrmodlist_add()` adds a new item consisting of the given modification operation code `mod_op` and attribute `attr` to the end of the modification list `modlist`. `attr`'s identifier indicates the attribute that is to be operated upon. `attr`'s values are used in a manner depending on the operation.

RETURN VALUES

`fn_attrmodlist_first()` returns 0 if the modification list is empty.

`fn_attrmodlist_next()` returns 0 if there are no more items on the modification list to be enumerated or if the iteration marker is invalid.

`fn_attrmodlist_add()` returns 1 if the operation succeeds, 0 if the operation fails.

USAGE Manipulation of attributes using the operations described in this manual page does not affect their representation in the underlying naming system. Changes to attributes in the underlying naming system can only be effected through the use of the interfaces described in `xfn_attributes(3N)`.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO `FN_attribute_t(3N)`, `FN_attrset_t(3N)`, `FN_identifier_t(3N)`, `fn_attr_modify(3N)`, `fn_attr_multi_modify(3N)`, `xfn(3N)`, `xfn_attributes(3N)`, `attributes(5)`

NOTES The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

| | |
|--------------------|--|
| NAME | fn_attr_multi_get, FN_multigetlist_t, fn_multigetlist_next, fn_multigetlist_destroy – return multiple attributes associated with named object |
| SYNOPSIS | <pre>cc [flag ...] file ... -lxfn [library ...] #include <xfn/xfn.h></pre> <pre>FN_multigetlist_t * fn_attr_multi_get(FN_ctx_t * ctx, const FN_composite_name_t * name, const FN_attrset_t * attr_ids, unsigned int follow_link, FN_status_t * status); FN_attribute_t * fn_multigetlist_next(FN_multigetlist_t * ml, FN_status_t * status); void fn_multigetlist_destroy(FN_multigetlist_t * ml, FN_status_t * status);</pre> |
| DESCRIPTION | <p>This set of operations returns one or more attributes associated with the object named by <i>name</i> relative to the context <i>ctx</i> . If <i>name</i> is empty, the attributes associated with <i>ctx</i> are returned.</p> <p>The value of <i>follow_link</i> determines what happens when the terminal atomic part of <i>name</i> is bound to an XFN link. If <i>follow_link</i> is non-zero, such a link is followed, and the values of the attribute associated with the final named object are returned; if <i>follow_link</i> is zero, such a link is not followed. Any XFN links encountered before the terminal atomic name are always followed.</p> <p>The attributes returned are those specified in <i>attr_ids</i> . If the value of <i>attr_ids</i> is 0 , all attributes associated with the named object are returned. Any attribute values in <i>attr_ids</i> provided by the caller are ignored; only the attribute identifiers are relevant for this operation. Each attribute (identifier, syntax, values) is returned one at a time using an enumeration scheme similar to that for listing a context.</p> <p>fn_attr_multi_get() initiates the enumeration process. It returns a handle to an FN_multigetlist_t object that can be used for the enumeration.</p> <p>The operation fn_multigetlist_next() returns a new FN_attribute_t object containing the next attribute (identifiers, syntaxes, and values) requested and updates <i>ml</i> to indicate the state of the enumeration.</p> |

RETURN VALUES

The operation **fn_multigetlist_destroy()** releases the resources used during the enumeration. It may be invoked before the enumeration has completed to terminate the enumeration.

fn_attr_multi_get() returns a pointer to an `FN_multigetlist_t` object if the enumeration has been initiated successfully; a `NULL` pointer (`0`) is returned if it failed.

fn_multigetlist_next() returns a pointer to an `FN_attribute_t` object if an attribute was returned, a `NULL` pointer (`0`) if no attribute was returned.

In the case of a failure, these operations set *status* to indicate the nature of the failure.

ERRORS

Each call to **fn_multigetlist_next()** sets *status* as follows:

| | |
|---|--|
| <code>FN_SUCCESS</code> | If an attribute was returned, there are more attributes to be enumerated. If no attribute was returned, the enumeration has completed successfully. |
| <code>FN_E_ATTR_NO_PERMISSION</code> | The caller did not have permission to read this attribute. |
| <code>FN_E_INSUFFICIENT_RESOURCES</code> | Insufficient resources are available to return the attribute's values. |
| <code>FN_E_INVALID_ATTR_IDENTIFIER</code> | This attribute identifier was not in a format acceptable to the naming system, or its contents was not valid for the format specified for the identifier. |
| <code>FN_E_INVALID_ENUM_HANDLE</code> | (No attribute should be returned with this status code). The given enumeration handle is not valid. Possible reasons could be that the handle was from another enumeration, or the object being processed no longer accepts the handle (due to such events as handle expiration or updates to the object's attribute set). |
| <code>FN_E_NO_SUCH_ATTRIBUTE</code> | The object did not have an attribute with the given identifier. |

FN_E_PARTIAL_RESULT (No attribute should be returned with this status code). The enumeration is not yet complete but cannot be continued.

For FN_E_ATTR_NO_PERMISSION, FN_E_INVALID_ATTR_IDENTIFIER, FN_E_INSUFFICIENT_RESOURCES, or FN_E_NO_SUCH_ATTRIBUTE, the returned attribute contains only the attribute identifier (no value or syntax). For these four status codes and FN_SUCCESS (when an attribute was returned), **fn_multigetlist_next()** can be called again to return another attribute. All other status codes indicate that no more attributes can be returned by **fn_multigetlist_next()** .

Other status codes, such as FN_E_COMMUNICATION_FAILURE, are also possible, in which case, no attribute is returned. In such cases, *status* is set as described in **FN_status_t(3N)** and **xfn_status_codes(3N)** .

USAGE

Implementations are not required to return all attributes requested by *attr_ids* . Some may choose to return only the attributes found successfully, followed by a status of FN_E_PARTIAL_RESULT; such implementations may not necessarily return attributes identifying those that could not be read. Implementations are not required to return the attributes in any order.

There may be a relationship between the *ctx* argument supplied to **fn_attr_multi_get()** and the FN_multigetlist_t object it returns. For example, some implementations may store the context handle *ctx* within the FN_multigetlist_t object for subsequent **fn_multigetlist_next()** calls. In general, a **fn_ctx_handle_destroy()** should not be invoked on *ctx* until the enumeration has terminated.

EXAMPLES

EXAMPLE 1 A sample program displaying how to use **fn_attr_multi_get()** function.

The following code fragment illustrates to obtain all attributes associated with a given name using the **fn_attr_multi_get()** operations.

```
/* list all attributes associated with given name */
extern FN_string_t *input_string;
FN_ctx_t *ctx;
FN_composite_name_t *target_name = fn_composite_name_from_string(input_string);
FN_multigetlist_t *ml;
FN_status_t *status = fn_status_create();
FN_attribute_t *attr;
int done = 0;
ctx = fn_ctx_handle_from_initial(status);
/* error checking on 'status' */
/* attr_ids == 0 indicates all attributes are to be returned */
if ((ml=fn_attr_multi_get(ctx, target_name, 0, status)) == 0) {
\011/* report 'status' and exit */
}
while ((attr=fn_multigetlist_next(ml, status)) && !done) {
\011switch (fn_status_code(status)) {
```

```

\011case FN_SUCCESS:
\011\011/* do something with 'attr' */
\011\011break;
\011case FN_E_ATTR_NO_PERMISSION:
\011case FN_E_ATTR_INVALID_ATTR_IDENTIFIER:
\011case FN_E_NO_SUCH_ATTRIBUTE:
\011\011/* report error using identifier in 'attr' */
\011\011break;
\011default:
\011\011/* other error handling */
\011\011done = 1;
\011}
\011if (attr)
\011\011fn_attribute_destroy(attr);
}
/* check 'status' for reason for end of enumeration and report if necessary */
/* clean up */
fn_multigetlist_destroy(ml, status);
/* report 'status' */

```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

FN_attribute_t(3N), **FN_attrset_t(3N)**, **FN_composite_name_t(3N)**, **FN_ctx_t(3N)**, **FN_identifier_t(3N)**, **FN_status_t(3N)**, **fn_attr_get(3N)**, **fn_ctx_handle_destroy(3N)**, **fn_ctx_list_names(3N)**, **xfn(3N)**, **xfn_attributes(3N)**, **xfn_status_codes(3N)**, **attributes(5)**

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

NAME | fn_attr_multi_modify – modify multiple attributes associated with named object

SYNOPSIS |

```
cc [ flag ... ] file ... -lxfn [ library ... ]
#include <xfn/xfn.h>
```

DESCRIPTION |

```
int fn_attr_multi_modify(FN_ctx_t *ctx, const FN_composite_name_t *name, const
FN_attrmodlist_t *mods, unsigned int follow_link, FN_attrmodlist_t **unexecuted_mods,
FN_status_t *status);
```

This operation modifies the attributes associated with the object named *name* relative to *ctx*. If *name* is empty, the attributes associated with *ctx* are modified.

The value of *follow_link* determines what happens when the terminal atomic part of *name* is bound to an XFN link. If *follow_link* is non-zero, such a link is followed, and the values of the attribute associated with the final named object are returned; if *follow_link* is zero, such a link is not followed. Any XFN links encountered before the terminal

In the *mods* parameter, the caller specifies a sequence of modifications that are to be done in order on the attributes. Each modification in the sequence specifies a modification operation code (see **fn_attr_modify(3N)**) and an attribute on which to operate.

The `FN_attrmodlist_t` type is described in **FN_attrmodlist_t(3N)**.

RETURN VALUES | **fn_attr_multi_modify()** returns 1 if all the modification operations were performed successfully. The function returns 0 if any error occurs. If the operation fails, *status* and *unexecuted_mods* are set as described below.

ERRORS | If an error is encountered while performing the list of modifications, *status* indicates the type of error and *unexecuted_mods* is set to a list of unexecuted modifications. The contents of *unexecuted_mods* do not share any state with *mods*; items in *unexecuted_mods* are copies of items in *mods* and appear in the same order in which they were originally supplied in *mods*. The first operation in *unexecuted_mods* is the first one that failed and the code in *status* applies to this modification operation in particular. If *status* indicates failure and a NULL pointer (0) is returned in *unexecuted_mods*, that indicates no modifications were executed.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | `FN_attrmodlist_t(3N)`, `FN_composite_name_t(3N)`, `FN_ctx_t(3N)`,
`FN_status_t(3N)`, `fn_attr_modify(3N)`, `xfn(3N)`,
`xfn_attributes(3N)`, `xfn_status_codes(3N)`, `attributes(5)`

NOTES | The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

| | |
|--------------------|--|
| NAME | fn_attr_search, FN_searchlist_t, fn_searchlist_next, fn_searchlist_destroy – search for the atomic name of objects with the specified attributes in a single context |
| SYNOPSIS | <pre>#include <xfn/xfn.h> FN_searchlist_t * fn_attr_search(FN_ctx_t * ctx, const FN_composite_name_t * name, const FN_attrset_t * match_attrs, unsigned int return_ref, const FN_attrset_t * return_attr_ids, FN_status_t * status); FN_string_t * fn_searchlist_next(FN_searchlist_t * sl, FN_ref_t ** returned_ref, FN_attrset_t ** returned_attrs, FN_status_t * status); void fn_searchlist_destroy(FN_searchlist_t * sl);</pre> |
| DESCRIPTION | <p>This set of operations is used to enumerate names of objects bound in the target context named <i>name</i> relative to the context <i>ctx</i> with attributes whose values match all those specified by <i>match_attrs</i> .</p> <p>The attributes specified by <i>match_attrs</i> form a conjunctive AND expression against which the attributes of each named object in the target context are evaluated. For multi-valued attributes, the list order of values is ignored and attribute values not specified in <i>match_attrs</i> are ignored. If no value is specified for an attribute in <i>match_attrs</i> , the presence of the attribute is tested. If the value of <i>match_attrs</i> is 0 , all names in the target context are enumerated.</p> <p>If a non-zero value of <i>return_ref</i> is passed to fn_attr_search() , the reference bound to the name is returned in the <i>returned_ref</i> argument to fn_searchlist_next() .</p> <p>Attribute identifiers and values associated with named objects that satisfy <i>match_attrs</i> may be returned by fn_searchlist_next() . The attributes returned are those listed in the <i>return_attr_ids</i> argument to fn_attr_search() . If the value of <i>return_attr_ids</i> is 0 , all attributes are returned. If <i>return_attr_ids</i> is an empty FN_attrset_t(3N) object, no attributes are returned. Any attribute values in <i>return_attr_ids</i> are ignored; only the attribute identifiers are relevant for <i>return_attr_ids</i> .</p> <p>The call to fn_attr_search() initiates the enumeration process. It returns a handle to an FN_searchlist_t object that is used to enumerate the names of the objects whose attributes match the attributes specified by <i>match_attrs</i> .</p> <p>The operation fn_searchlist_next() returns the next name in the enumeration identified by the <i>sl</i> . The reference of the name is returned in <i>returned_ref</i> if <i>return_ref</i> was set in the call to fn_attr_search() . The attributes specified by <i>return_attr_ids</i> are returned in <i>returned_attrs</i> . fn_searchlist_next() also updates <i>sl</i> to indicate the state of the enumeration. Successive calls to</p> |

fn_searchlist_next() using *sl* return successive names, and optionally, references and attributes, in the enumeration; these calls further update the state of the enumeration.

fn_searchlist_destroy() releases resources used during the enumeration. This can be invoked at any time to terminate the enumeration.

fn_attr_search() does not follow XFN links that are bound in the target context.

RETURN VALUES

fn_attr_search() returns a pointer to an `FN_searchlist_t` object if the enumeration is successfully initiated; it returns a `NULL` pointer if the enumeration cannot be initiated or if no named object with attributes whose values match those specified in *match_attrs* is found.

fn_searchlist_next() returns a pointer to an `FN_string_t(3N)` object; it returns a `NULL` pointer if no more names can be returned in the enumeration. If *returned_ref* is a `NULL` pointer, or if the *return_ref* parameter to *fn_attr_search* was 0, no reference is returned; otherwise, *returned_ref* contains the reference bound to the name. If *returned_attrs* is a `NULL` pointer, no attributes are returned; otherwise, *returned_attrs* contains the attributes associated with the named object, as specified by the *return_attr_ids* parameter to **fn_attr_search()**.

In the case of a failure, these operations return in the *status* argument a code indicating the nature of the failure.

ERRORS

fn_attr_search() returns a `NULL` pointer if the enumeration could not be initiated. The *status* argument is set in the following way:

| | |
|--------------------------------------|--|
| <code>FN_SUCCESS</code> | A named object could not be found whose attributes satisfied the implied filter of equality and conjunction. |
| <code>FN_E_ATTR_NO_PERMISSION</code> | The caller did not have permission to read one or more of the specified attributes. |
| <code>FN_E_INVALID_ATTR_VALUE</code> | A value type in the specified attributes did not match the syntax of the attribute against which it was being evaluated. |

Other status codes are possible as described in `FN_status_t(3N)` and `xfn_status_codes(3N)`.

Each successful call to **fn_searchlist_next()** returns a name and, optionally, the reference and requested attributes. *status* is set in the following way:

| | |
|-------------------------|--|
| <code>FN_SUCCESS</code> | All requested attributes were returned successfully with the name. |
|-------------------------|--|

| | |
|---|--|
| FN_E_ATTR_NO_PERMISSION | The caller did not have permission to read one or more of the requested attributes. |
| FN_E_INVALID_ATTR_IDENTIFIER | A requested attribute identifier was not in a format acceptable to the naming system, or its contents was not valid for the format specified. |
| FN_E_NO_SUCH_ATTRIBUTE | The named object did not have one of the requested attributes. |
| FN_E_INSUFFICIENT_RESOURCES | Insufficient resources are available to return all the requested attributes and their values. |
| FN_E_ATTR_NO_PERMISSION FN_E_INVALID_ATTR_IDENTIFIER FN_E_NO_SUCH_ATTRIBUTE FN_E_INSUFFICIENT_RESOURCES | These indicate that some of the requested attributes may have been returned in <i>returned_attrs</i> but one or more of them could not be returned. Use <code>fn_attr_get(3N)</code> or <code>fn_attr_multi_get(3N)</code> to discover why these attributes could not be returned. |
| fn_searchlist_next() returns a NULL pointer if no more names can be returned. The status argument is set in the following way: | |
| FN_SUCCESS | The search has completed successfully. |
| FN_E_PARTIAL_RESULT | The enumeration is not yet complete but cannot be continued. |
| FN_E_ATTR_NO_PERMISSION | The caller did not have permission to read one or more of the specified attributes. |
| FN_E_INVALID_ENUM_HANDLE | The supplied enumeration handle was not valid. Possible reasons could be that the handle was from another enumeration, or the context being enumerated no longer accepts the |

handle (due to such events as handle expiration or updates to the context).

Other status codes are possible as described in `FN_status_t(3N)` and `xfn_status_codes(3N)`.

USAGE

The names enumerated using `fn_searchlist_next()` are not ordered in any way. Furthermore, there is no guarantee that any two series of enumerations on the same context with identical `match_attrs` will return the names in the same order.

EXAMPLES

EXAMPLE 1 A sample program of displaying how to use `fn_attr_search()` function.

The following code fragment illustrates how the `fn_attr_search()` operation may be used. The code consists of three parts: preparing the arguments for the search, performing the search, and cleaning up.

The first part involves getting the name of the context to start the search and constructing the set of attributes that named objects in the context must satisfy. This is done in the declarations part of the code and by the routine `get_search_query`.

The next part involves doing the search and enumerating the results of the search. This is done by first getting a context handle to the Initial Context, and then passing that handle along with the name of the target context and matching attributes to `fn_attr_search()`. This particular call to `fn_attr_search()` is requesting that no reference be returned (by passing in 0 for `return_ref`), and that all attributes associated with the named object be returned (by passing in 0 as the `return_attr_ids` argument). If successful, `fn_attr_search()` returns `sl`, a handle for enumerating the results of the search. The results of the search are enumerated using calls to `fn_searchlist_next()`, which returns the name of the object and the attributes associated with the named object in `returned_attrs`.

The last part of the code involves cleaning up the resources used during the search and enumeration. The call to `fn_searchlist_destroy()` releases resources reserved for this enumeration. The other calls release the context handle, name, attribute set, and status objects created earlier.

```

/* Declarations */
FN_ctx_t *ctx;
FN_searchlist_t *sl;
FN_string_t *name;
FN_attrset_t *returned_attrs;
FN_status_t *status = fn_status_create();
FN_composite_name_t *target_name = get_name_from_user_input();
FN_attrset_t *match_attrs = get_search_query();
/* Get context handle to Initial Context */
ctx = fn_ctx_handle_from_initial(status);
/* error checking on 'status' */
/* Initiate search */
if ((sl=fn_attr_search(ctx, target_name, match_attrs,
\011/* no reference */ 0, /* return all attrs */ 0, status)) == 0) {

```

```

\011/* report 'status', cleanup, and exit */
}
/* Enumerate names and attributes requested */
while (name=fn_searchlist_next(sl, 0, &returned_attrs, status)) {
\011/* do something with 'name' and 'returned_attrs' */
\011fn_string_destroy(name);
\011fn_attrset_destroy(returned_attrs);
}
/* check 'status' for reason for end of enumeration */
/* Clean up */
fn_searchlist_destroy(sl); /* Free resources of 'sl' */
fn_status_destroy(status);
fn_attrset_destroy(match_attrs);
fn_ctx_handle_destroy(ctx);
fn_composite_name_destroy(target_name);
/*
 * Procedure for constructing attribute set containing
 * attributes to be matched:
 * \011"zip_code" attribute value is "02158"
 * \011AND "employed" attribute is present.
 */
FN_attrset_t *
get_search_query()
{
\011/* Zip code and employed attribute identifier, syntax */
\011extern FN_attribute_t\011\011 *attr_zip_code;
\011extern FN_attribute_t\011\011 *attr_employed;
\011FN_attribute_t *zip_code = fn_attribute_copy(attr_zip_code);
\011FN_attr_value_t zc_value = {5, "02158"};
\011FN_attrset_t *match_attrs = fn_attrset_create();
\011fn_attribute_add(zip_code, &zc_value, 0);
\011fn_attrset_add(match_attrs, zip_code, 0);
\011fn_attrset_add(match_attrs, attr_employed, 0);
\011return (match_attrs);
}

```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

FN_attribute_t(3N), **FN_attrset_t(3N)**, **FN_attrvalue_t(3N)**, **FN_composite_name_t(3N)**, **FN_ctx_t(3N)**, **FN_status_t(3N)**, **FN_string_t(3N)**, **fn_attr_ext_search(3N)**, **fn_attr_get(3N)**, **fn_attr_multi_get(3N)**, **fn_ctx_list_names(3N)**, **xfn_status_codes(3N)**, **attributes(5)**

| | |
|--------------------|---|
| NAME | FN_attrset_t, fn_attrset_create, fn_attrset_destroy, fn_attrset_copy, fn_attrset_assign, fn_attrset_get, fn_attrset_count, fn_attrset_first, fn_attrset_next, fn_attrset_add, fn_attrset_remove – a set of XFN attributes |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lxfn [<i>library</i> ...] #include <xfn/xfn.h> FN_attrset_t * fn_attrset_create(void); void fn_attrset_destroy(FN_attrset_t * aset); FN_attrset_t * fn_attrset_copy(const FN_attrset_t * aset); FN_attrset_t * fn_attrset_assign(FN_attrset_t * dst, const FN_attrset_t * src); const FN_attribute_t * fn_attrset_get(const FN_attrset_t * aset, const FN_identifier_t * attr_id); unsigned int fn_attrset_count(const FN_attrset_t * aset); const FN_attribute_t * fn_attrset_first(const FN_attrset_t * aset, void ** iter_pos); const FN_attribute_t * fn_attrset_next(const FN_attrset_t * aset, void ** iter_pos); int fn_attrset_add(FN_attrset_t * aset, const FN_attribute_t * attr, unsigned int exclusive); int fn_attrset_remove(FN_attrset_t * aset, const FN_identifier_t * attr_id); </pre> |
| DESCRIPTION | <p>An attribute set is a set of attribute objects with distinct identifiers. The fn_attr_multi_get(3N) operation takes an attribute set as parameter and returns an attribute set. The fn_attr_get_ids(3N) operation returns an attribute set containing the identifiers of the attributes.</p> <p>Attribute sets are represented by the type <code>FN_attrset_t</code>. The following operations are defined for manipulating attribute sets.</p> <p>fn_attrset_create() creates an empty attribute set. fn_attrset_destroy() releases the storage associated with the attribute set <code>aset</code>. fn_attrset_copy() returns a copy of the attribute set <code>aset</code>. fn_attrset_assign() makes a copy of the</p> |

attribute set *src* and assigns it to *dst* , releasing any old contents of *dst* . A pointer to the same object as *dst* is returned.

fn_attrset_get() returns the attribute with the given identifier *attr_id* from *aset* . **fn_attrset_count()** returns the number attributes found in the attribute set *aset* .

fn_attrset_first() and **fn_attrset_next()** are functions that can be used to return an enumeration of all the attributes in an attribute set. The attributes are not ordered in any way. There is no guaranteed relation between the order in which items are added to an attribute set and the order of the enumeration. The specification does guarantee that any two enumerations will return the members in the same order, provided that no **fn_attrset_add()** or **fn_attrset_remove()** operation was performed on the object in between or during the two enumerations. **fn_attrset_first()** returns the first attribute from the set and sets *iter_pos* after the first attribute. **fn_attrset_next ()** returns the attribute following *iter_pos* and advances *iter_pos* .

fn_attrset_add() adds the attribute *attr* to the attribute set *aset* , replacing the attribute's values if the identifier of *attr* is not distinct in *aset* and *exclusive* is 0 . If *exclusive* is non-zero and the identifier of *attr* is not distinct in *aset* , the operation fails.

fn_attrset_remove() removes the attribute with the identifier *attr_id* from *aset* . The operation succeeds even if no such attribute occurs in *aset* .

RETURN VALUES

fn_attrset_first() returns 0 if the attribute set is empty. **fn_attrset_next()** returns 0 if there are no more attributes in the set.

fn_attrset_add() and **fn_attrset_remove()** return 1 if the operation succeeds, and 0 if the operation fails.

USAGE

Manipulation of attributes using the operations described in this manual page does not affect their representation in the underlying naming system. Changes to attributes in the underlying naming system can only be effected through the use of the interfaces described in **xfn_attributes(3N)** .

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

FN_attribute_t(3N) , **FN_attrvalue_t(3N)** , **FN_identifier_t(3N)** , **fn_attr_get_ids(3N)** , **fn_attr_multi_get(3N)** , **xfn(3N)** , **xfn_attributes(3N)** , **attributes(5)**

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

| | |
|--------------------|--|
| NAME | FN_attrvalue_t - an XFN attribute value |
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lxfn [<i>library ...</i>] #include <xfn/xfn.h></pre> |
| DESCRIPTION | <p>The type FN_attrvalue_t is used to represent the contents of a single attribute value, within an attribute of type FN_attribute_t.</p> <p>The representation of this structure is defined by XFN as follows:</p> <pre>typedef struct { size_t <i>length</i>; void *<i>contents</i>; } FN_attrvalue_t;</pre> |
| SEE ALSO | FN_attribute_t(3N), fn_attr_get_values(3N), xfn(3N) |

NAME FN_composite_name_t, fn_composite_name_create, fn_composite_name_destroy, fn_composite_name_from_str, fn_composite_name_from_string, fn_string_from_composite_name, fn_composite_name_copy, fn_composite_name_assign, fn_composite_name_is_empty, fn_composite_name_count, fn_composite_name_first, fn_composite_name_next, fn_composite_name_prev, fn_composite_name_last, fn_composite_name_prefix, fn_composite_name_suffix, fn_composite_name_is_equal, fn_composite_name_is_prefix, fn_composite_name_is_suffix, fn_composite_name_prepend_comp, fn_composite_name_append_comp, fn_composite_name_insert_comp, fn_composite_name_delete_comp, fn_composite_name_prepend_name, fn_composite_name_append_name, fn_composite_name_insert_name – a sequence of component names spanning multiple naming systems

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lxfn
[
library
... ]
#include <xfn/xfn.h>

FN_composite_name_t * fn_composite_name_create(void);

void fn_composite_name_destroy(FN_composite_name_t * name);

FN_composite_name_t * fn_composite_name_from_str(const unsigned char * cstr);

FN_composite_name_t * fn_composite_name_from_string(const FN_string_t * str);

FN_string_t * fn_string_from_composite_name(const FN_composite_name_t *
name, unsigned int * status);

FN_composite_name_t * fn_composite_name_copy(const FN_composite_name_t *
name);

FN_composite_name_t * fn_composite_name_assign(FN_composite_name_t * dst,
const FN_composite_name_t * src);

int fn_composite_name_is_empty(const FN_composite_name_t * name);

unsigned int fn_composite_name_count(const FN_composite_name_t * name);
```

```

const FN_string_t * fn_composite_name_first(const FN_composite_name_t * name,
void ** iter_pos);

const FN_string_t * fn_composite_name_next(const FN_composite_name_t * name,
void ** iter_pos);

const FN_string_t * fn_composite_name_prev(const FN_composite_name_t * name,
void ** iter_pos);

const FN_string_t * fn_composite_name_last(const FN_composite_name_t * name,
void ** iter_pos);

FN_composite_name_t * fn_composite_name_prefix(const FN_composite_name_t *
name, const void * iter_pos);

FN_composite_name_t * fn_composite_name_suffix(const FN_composite_name_t *
name, const void * iter_pos);

int fn_composite_name_is_equal(const FN_composite_name_t * name, const
FN_composite_name_t * name2, unsigned int * status);

int fn_composite_name_is_prefix(const FN_composite_name_t * name, const
FN_composite_name_t * prefix, void ** iter_pos, unsigned int * status);

int fn_composite_name_is_suffix(const FN_composite_name_t * name, const
FN_composite_name_t * suffix, void ** iter_pos, unsigned int * status);

int fn_composite_name_prepend_comp(FN_composite_name_t * name, const
FN_string_t * newcomp);

int fn_composite_name_append_comp(FN_composite_name_t * name, const
FN_string_t * newcomp);

int fn_composite_name_insert_comp(FN_composite_name_t * name, void **
iter_pos, const FN_string_t * newcomp);

int fn_composite_name_delete_comp(FN_composite_name_t * name, void **
iter_pos);

int fn_composite_name_prepend_name(FN_composite_name_t * name, const
FN_composite_name_t * newcomps);

int fn_composite_name_append_name(FN_composite_name_t * name, const
FN_composite_name_t * newcomps);

int fn_composite_name_insert_name(FN_composite_name_t * name, void **
iter_pos, const FN_composite_name_t * newcomps);

```

DESCRIPTION

A composite name is represented by an object of type `FN_composite_name_t`. Each component is a string name, of type

`FN_string_t`, from the namespace of a single naming system. It may be an atomic name or a compound name in that namespace.

`fn_composite_name_create` creates an `FN_composite_name_t` object with zero components. Components may be subsequently added to the composite name using the modify operations described below.

`fn_composite_name_destroy` releases any storage associated with the given `FN_composite_name_t` handle.

`fn_composite_name_from_str()` creates an `FN_composite_name_t` from the given null-terminated string based on the code set of the current locale setting, using the XFN composite name syntax. `fn_composite_name_from_string()` creates an `FN_composite_name_t` from the string `str` using the XFN composite name syntax. `fn_string_from_composite_name()` returns the standard string form of the given composite name, by concatenating the components of the composite name in a left to right order, each separated by the XFN component separator.

`fn_composite_name_copy()` returns a copy of the given composite name object. `fn_composite_name_assign()` makes a copy of the composite name object pointed to by `src` and assigns it to `dst`, releasing any old contents of `dst`. A pointer to the same object as `dst` is returned.

`fn_composite_name_is_empty()` returns 1 if the given composite name is an empty composite name (that is, it consists of a single, empty component name); otherwise, it returns 0. `fn_composite_name_count()` returns the number of components in the given composite name.

The iteration scheme is based on the exchange of an opaque `void *` argument, `iter_pos`, that serves to record the position of the iteration in the sequence. Conceptually, `iter_pos` records a position between two successive components (or at one of the extreme ends of the sequence).

The function `fn_composite_name_first()` returns a handle to the `FN_string_t` that is the first component in the name, and sets `iter_pos` to indicate the position immediately following the first component. It returns 0 if the name has no components. Thereafter, successive calls of the `fn_composite_name_next()` function return pointers to the component following the iteration marker, and advance the iteration marker. If the iteration marker is at the end of the sequence, `fn_composite_name_next()` returns 0. Similarly,

`fn_composite_name_prev()` returns the component preceding the iteration pointer and moves the marker back one component. If the marker is already at the beginning of the sequence, `fn_composite_name_prev()` returns 0. The function `fn_composite_name_last()` returns a pointer to the last component of the name and sets the iteration marker immediately preceding this component (so that subsequent calls to `fn_composite_name_prev()` can be used to step through leading components of the name).

The **fn_composite_name_suffix()** function returns a composite name consisting of a copy of those components following the supplied iteration marker. The method **fn_composite_name_prefix()** returns a composite name consisting of those components that precede the iteration marker. Using these functions with an iteration marker that was not initialized using **fn_composite_name_first()** , **fn_composite_name_last()** , **fn_composite_name_is_prefix()** , or **fn_composite_name_is_suffix()** yields undefined and generally undesirable behavior.

The functions **fn_composite_name_is_equal()** , **fn_composite_name_is_prefix()** , and **fn_composite_name_is_suffix()** test for equality between composite names or between parts of composite names. For these functions, equality is defined as exact string equality, not name equivalence. A name's syntactic property, such as case-insensitivity, is not taken into account by these functions.

The function **fn_composite_name_is_prefix()** tests if one composite name is a prefix of another. If so, it returns 1 and sets the iteration marker immediately following the prefix. (For example, a subsequent call to **fn_composite_name_suffix()** will return the remainder of the name.) Otherwise, it returns 0 and the value of the iteration marker is undefined. The function **fn_composite_name_is_suffix()** is similar. It tests if one composite name is a suffix of another. If so, it returns 1 and sets the iteration marker immediately preceding the suffix.

The functions **fn_composite_name_prepend_comp()** and **fn_composite_name_append_comp()** prepend and append a single component to the given composite name, respectively. These operations invalidate any iteration marker the client holds for that object.

fn_composite_name_insert_comp() inserts a single component before *iter_pos* to the given composite name and sets *iter_pos* to be immediately after the component just inserted. **fn_composite_name_delete_comp()** deletes the component located before *iter_pos* from the given composite name and sets *iter_pos* back one component.

The functions **fn_composite_name_prepend_name()** , **fn_composite_name_append_name()** , and **fn_composite_name_insert_name()** perform the same update functions as their *_comp* counterparts, respectively, except that multiple components are being added, rather than single components. For example, **fn_composite_name_insert_name()** sets *iter_pos* to be immediately after the name just added.

RETURN VALUES

The functions **fn_composite_name_is_empty()** , **fn_composite_name_is_equal()** , **fn_composite_name_is_suffix()** , and **fn_composite_name_is_prefix()** return 1 if the test indicated is true; 0 otherwise.

The update functions **fn_composite_name_prepend_comp()** , **fn_composite_name_append_comp()** , **fn_composite_name_insert_comp()** , **fn_composite_name_delete_comp()** , and their *_name* counterparts return 1 if the update was successful; 0 otherwise.

If a function is expected to return a pointer to an object, a NULL pointer (0) is returned if the function fails.

ERRORS

Code set mismatches that occur during the composition of the string form or during comparisons of composite names are resolved in an implementation-dependent way. **fn_string_from_composite_name()** , **fn_composite_name_is_equal()** , **fn_composite_name_is_suffix()** , and **fn_composite_name_is_prefix()** set *status* to **FN_E_INCOMPATIBLE_CODE_SETS** for composite names whose components have code sets that are determined by the implementation to be incompatible.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

FN_string_t(3N) , **xfn(3N)** , **attributes(5)**

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

NAME FN_compound_name_t, fn_compound_name_from_syntax_attrs, fn_compound_name_get_syntax_attrs, fn_compound_name_destroy, fn_string_from_compound_name, fn_compound_name_copy, fn_compound_name_assign, fn_compound_name_count, fn_compound_name_first, fn_compound_name_next, fn_compound_name_prev, fn_compound_name_last, fn_compound_name_prefix, fn_compound_name_suffix, fn_compound_name_is_empty, fn_compound_name_is_equal, fn_compound_name_is_prefix, fn_compound_name_is_suffix, fn_compound_name_prepend_comp, fn_compound_name_append_comp, fn_compound_name_insert_comp, fn_compound_name_delete_comp, fn_compound_name_delete_all – an XFN compound name

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lxfn
[
library
... ]
#include <xfn/xfn.h>

FN_compound_name_t * fn_compound_name_from_syntax_attrs(const
FN_attrset_t * aset, const FN_string_t * name, FN_status_t * status);

FN_attrset_t * fn_compound_name_get_syntax_attrs(const FN_compound_name_t
* name);

void fn_compound_name_destroy(FN_compound_name_t * name);

FN_string_t * fn_string_from_compound_name(const FN_compound_name_t *
name);

FN_compound_name_t * fn_compound_name_copy(const FN_compound_name_t *
name);

FN_compound_name_t * fn_compound_name_assign(FN_compound_name_t * dst,
const FN_compound_name_t * src);

unsigned int fn_compound_name_count(const FN_compound_name_t * name);

const FN_string_t * fn_compound_name_first(const FN_compound_name_t * name,
void ** iter_pos);
```

```

const FN_string_t * fn_compound_name_next(const FN_compound_name_t * name,
void ** iter_pos);

const FN_string_t * fn_compound_name_prev(const FN_compound_name_t * name,
void ** iter_pos);

const FN_string_t * fn_compound_name_last(const FN_compound_name_t * name,
void ** iter_pos);

FN_compound_name_t * fn_compound_name_prefix(const FN_compound_name_t *
name, const void * iter_pos);

FN_compound_name_t * fn_compound_name_suffix(const FN_compound_name_t *
name, const void * iter_pos);

int fn_compound_name_is_empty(const FN_compound_name_t * name);

int fn_compound_name_is_equal(const FN_compound_name_t * name1, const
FN_compound_name_t * name2, unsigned int * status);

int fn_compound_name_is_prefix(const FN_compound_name_t * name, const
FN_compound_name_t * pre, void ** iter_pos, unsigned int * status);

int fn_compound_name_is_suffix(const FN_compound_name_t * name, const
FN_compound_name_t * suffix, void ** iter_pos, unsigned int * status);

int fn_compound_name_prepend_comp(FN_compound_name_t * name, const
FN_string_t * atomic_comp, unsigned int * status);

int fn_compound_name_append_comp(FN_compound_name_t * name, const
FN_string_t * atomic_comp, unsigned int * status);

int fn_compound_name_insert_comp(FN_compound_name_t * name, void ** iter_pos,
const FN_string_t * atomic_comp, unsigned int * status);

int fn_compound_name_delete_comp(FN_compound_name_t * name, void **
iter_pos);

int fn_compound_name_delete_all(FN_compound_name_t * name);

```

DESCRIPTION

Most applications treat names as opaque data. Hence, the majority of clients of the XFN interface will not need to parse names. Some applications, however, such as browsers, need to parse names. For these applications, XFN provides support in the form of the `FN_compound_name_t` object.

Each naming system in an XFN federation potentially has its own naming conventions. The `FN_compound_name_t` object has associated operations for applications to process compound names that conform to the XFN model of expressing compound name syntax. The XFN syntax model for compound names covers a large number of specific name syntaxes and is expressed in

terms of syntax properties of the naming convention. See **xfn_compound_names(3N)**.

An **FN_compound_name_t** object is constructed by the operation **fn_compound_name_from_syntax_attrs**, using a string name and an attribute set containing the "fn_syntax_type" (with identifier format **FN_ID_STRING**) attribute identifying the namespace syntax of the string name. The value "standard" (with identifier format **FN_ID_STRING**) in the "fn_syntax_type" specifies a syntax model that is by default supported by the **FN_compound_name_t** object. An implementation may support other syntax types instead of the XFN standard syntax model, in which case the value of the "fn_syntax_type" attribute would be set to an implementation-specific string. **fn_compound_name_get_syntax_attrs()** returns an attribute set containing the syntax attributes that describes the given compound name. **fn_compound_name_destroy()** releases the storage associated with the given compound name. **fn_string_from_compound_name()** returns the string form of the given compound name. **fn_compound_name_copy()** returns a copy of the given compound name. **fn_compound_name_assign()** makes a copy of the compound name *src* and assigns it to *dst*, releasing any old contents of *dst*. A pointer to the object pointed to by *dst* is returned. **fn_compound_name_count()** returns the number of atomic components in the given compound name.

The function **fn_compound_name_first()** returns a handle to the **FN_string_t** that is the first atomic component in the compound name, and sets *iter_pos* to indicate the position immediately following the first component. It returns 0 if the name has no components. Thereafter, successive calls of the **fn_compound_name_next()** function return pointers to the component following the iteration marker, and advance the iteration marker. If the iteration marker is at the end of the sequence, **fn_compound_name_next()** returns 0. Similarly, **fn_compound_name_prev()** returns the component preceding the iteration pointer and moves the marker back one component. If the marker is already at the beginning of the sequence, **fn_compound_name_prev()** returns 0. The function **fn_compound_name_last()** returns a pointer to the last component of the name and sets the iteration marker immediately preceding this component (so that subsequent calls to **fn_compound_name_prev()** can be used to step through trailing components of the name).

The **fn_compound_name_suffix()** function returns a compound name consisting of a copy of those components following the supplied iteration marker. The function **fn_compound_name_prefix()** returns a compound name consisting of those components that precede the iteration marker. Using these functions with an iteration marker that was not initialized with the use of **fn_compound_name_first()**, **fn_compound_name_last()**, **fn_compound_name_is_prefix()**, or **fn_compound_name_is_suffix()** yields undefined and generally undesirable behavior.

The functions **fn_compound_name_is_equal()** , **fn_compound_name_is_prefix()** , and **fn_compound_name_is_suffix()** test for equality between compound names or between parts of compound names. For these functions, equality is defined as name equivalence. A name's syntactic property, such as case-insensitivity, is taken into account by these functions.

The function **fn_compound_name_is_prefix()** tests if one compound name is a prefix of another. If so, it returns 1 and sets the iteration marker immediately following the prefix. (For example, a subsequent call to **fn_compound_name_suffix()** will return the remainder of the name.) Otherwise, it returns 0 and value of the iteration marker is undefined. The function **fn_compound_name_is_suffix()** is similar. It tests if one compound name is a suffix of another. If so, it returns 1 and sets the iteration marker immediately preceding the suffix.

The functions **fn_compound_name_prepend_comp()** and **fn_compound_name_append_comp()** prepend and append a single atomic component to the given compound name, respectively. These operations invalidate any iteration marker the client holds for that object. **fn_compound_name_insert_comp()** inserts an atomic component before *iter_pos* to the given compound name and sets *iter_pos* to be immediately after the component just inserted. **fn_compound_name_delete_comp()** deletes the atomic component located before *iter_pos* from the given compound name and sets *iter_pos* back one component. **fn_compound_name_delete_all()** deletes all the atomic components from *name* .

RETURN VALUES

The following test functions return 1 if the test indicated is true; otherwise, they return 0 :

fn_compound_name_is_empty()
fn_compound_name_is_equal()
fn_compound_name_is_suffix()
fn_compound_name_is_prefix()

The following update functions return 1 if the update was successful; otherwise, they return 0 :

fn_compound_name_prepend_comp()
fn_compound_name_append_comp()
fn_compound_name_insert_comp()
fn_compound_name_delete_comp()
fn_compound_name_delete_all()

If a function is expected to return a pointer to an object, a NULL pointer (0) is returned if the function fails.

ERRORS

When the function **fn_compound_name_from_syntax_attrs()** fails, it returns a status code in *status* . The possible status codes are:

- FN_E_ILLEGAL_NAME The name supplied to the operation was not a well- formed XFN compound name, or one of the component names was not well-formed according to the syntax of the naming system(s) involved in its resolution.
- FN_E_INCOMPATIBLE_CODE_SETS The code set of the given string is incompatible with that supported by the compound name.
- FN_E_INVALID_SYNTAX_ATTRS The syntax attributes supplied are invalid or insufficient to fully specify the syntax.
- FN_E_SYNTAX_NOT_SUPPORTED The syntax type specified is not supported.

The following functions may return in *status* the status code FN_E_INCOMPATIBLE_CODE_SETS when the code set of the given string is incompatible with that of the compound name:

- fn_compound_name_is_equal()**
- fn_compound_name_is_suffix()**
- fn_compound_name_is_prefix()**
- fn_compound_name_prepend_comp()**
- fn_compound_name_append_comp()**
- fn_compound_name_insert_comp()**

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | `FN_attribute_t(3N)`, `FN_attrset_t(3N)`, `FN_composite_name_t(3N)`, `FN_status_t(3N)`, `FN_string_t(3N)`, `fn_ctx_get_syntax_attrs(3N)`, `xfn(3N)`, `xfn_compound_names(3N)`, `attributes(5)`

NOTES | The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

| NAME | fn_ctx_bind – bind a reference to a name | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lxfn [library ...] #include <xfn/xfn.h></pre> <p>int fn_ctx_bind(FN_ctx_t *ctx, const FN_composite_name_t *name, const FN_ref_t *ref, unsigned int exclusive, FN_status_t *status);</p> | | | | |
| DESCRIPTION | <p>This operation binds the supplied reference <i>ref</i> to the supplied composite name <i>name</i> relative to <i>ctx</i>. The binding is made in the target context, that is, the context named by all but the terminal atomic part of <i>name</i>. The operation binds the terminal atomic name to the supplied reference in the target context. The target context must already exist.</p> <p>The value of <i>exclusive</i> determines what happens if the terminal atomic part of the name is already bound in the target context. If <i>exclusive</i> is nonzero and <i>name</i> is already bound, the operation fails. If <i>exclusive</i> is 0, the new binding replaces any existing binding.</p> | | | | |
| RETURN VALUES | When the bind operation is successful it returns 1; on error it returns 0. | | | | |
| ERRORS | fn_ctx_bind sets <i>status</i> as described in FN_status_t(3N) and xfn_status_codes . Of special relevance for this operation is the status code FN_E_NAME_IN_USE , which indicates that the supplied name is already in use. | | | | |
| USAGE | <p>The value of <i>ref</i> cannot be NULL. If the intent is to reserve a name using fn_ctx_bind(), a reference containing no address should be supplied. This reference may be name service-specific or it may be the conventional NULL reference defined in the X/Open registry (see fns_references(5)).</p> <p>If multiple sources are updating a reference, they must synchronize amongst each other when adding, modifying, or removing from the address list of a bound reference.</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | FN_composite_name_t(3N) , FN_ctx_t(3N) , FN_ref_t(3N) , FN_status_t(3N) , fn_ctx_lookup(3N) , fn_ctx_unbind(3N) , xfn(3N) , xfn_status_codes(3N) , attributes(5) , fns_references(5) | | | | |

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

NAME | fn_ctx_create_subcontext – create a subcontext in a context

SYNOPSIS |

```
cc [ flag ... ] file ... -lxfn [ library ... ]
#include <xfn/xfn.h>
```

FN_ref_t *fn_ctx_create_subcontext(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);

DESCRIPTION | This operation creates a new XFN context of the same type as the target context — that named by all but the terminal atomic component of *name* — and binds it to the supplied composite name.

As with `fn_ctx_bind()`, the target context must already exist. The new context is created and bound in the target context using the terminal atomic name in *name*. The operation returns a reference to the newly created context.

RETURN VALUE | `fn_ctx_create_subcontext()` returns a reference to the newly created context; if the operation fails, it returns a NULL pointer (0).

ERRORS | `fn_ctx_create_subcontext()` sets *status* as described in `FN_status_t(3N)` and `xfn_status_codes(3N)`. Of special relevance for this operation is the following status code:

| | |
|-------------------------------|--|
| <code>FN_E_NAME_IN_USE</code> | The terminal atomic name already exists in the target context. |
|-------------------------------|--|

APPLICATION USAGE | The new subcontext is an XFN context and is created in the same naming system as the target context. The new subcontext also inherits the same syntax attributes as the target context. XFN does not specify any further properties of the new subcontext. The target context and its naming system determine these.

ATTRIBUTES | See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe. |

SEE ALSO | `FN_composite_name_t(3N)`, `FN_ctx_t(3N)`, `FN_ref_t(3N)`, `FN_status_t(3N)`, `fn_ctx_bind(3N)`, `fn_ctx_lookup(3N)`, `fn_ctx_destroy_subcontext(3N)`, `xfn_status_codes(3N)`, `xfn(3N)`, `attributes(5)`

| | |
|--------------------------|---|
| NAME | fn_ctx_destroy_subcontext – destroy the named context and remove its binding from the parent context |
| SYNOPSIS | <pre>cc [flag ...] file ... -lxfn [library ...] #include <xfn/xfn.h></pre> <pre>int fn_ctx_destroy_subcontext(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);</pre> |
| DESCRIPTION | <p>This operation destroys the subcontext named by <i>name</i> relative to <i>ctx</i>, and unbinds the name.</p> <p>As with <code>fn_ctx_unbind()</code>, this operation succeeds even if the terminal atomic name is not bound in the target context — the context named by all but the terminal atomic name in <i>name</i>.</p> |
| RETURN VALUE | <code>fn_ctx_destroy_subcontext()</code> returns 1 on success and 0 on failure. |
| ERRORS | <p><code>fn_ctx_destroy_subcontext()</code> sets <i>status</i> as described in <code>FN_status_t(3N)</code> and <code>xfn_status_codes(3N)</code>. Of special relevance for <code>fn_ctx_destroy_subcontext()</code> are the following status codes:</p> <p><code>FN_E_CTX_NOT_A_CONTEXT</code> <i>name</i> does not name a context.</p> <p><code>FN_E_CTX_NOT_EMPTY</code> The naming system being asked to do the destroy does not support removal of a context that still contains bindings.</p> |
| APPLICATION USAGE | <p>Some aspects of this operation are not specified by XFN, but are determined by the target context and its naming system. For example, XFN does not specify what happens if the named subcontext is non-empty when the operation is invoked.</p> <p>In naming systems that support attributes, and store the attributes along with names or contexts, this operation removes the name, the context, and its associated attributes.</p> <p>Normal resolution always follows links. In a <code>fn_ctx_destroy_subcontext()</code> operation, resolution of <i>name</i> continues to the target context; the terminal atomic name is not resolved. If the terminal atomic name is bound to a link, the link is not followed and the operation fails with <code>FN_E_CTX_NOT_A_CONTEXT</code> because the name is not bound to a context.</p> |
| ATTRIBUTES | See <code>attributes(5)</code> for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe. |

SEE ALSO

`FN_ctx_t(3N)`, `FN_composite_name_t(3N)`, `FN_status_t(3N)`,
`fn_ctx_create_subcontext(3N)`, `fn_ctx_unbind(3N)`, `xfn(3N)`,
`xfn_status_codes(3N)`, `attributes(5)`

| | | | |
|--------------------------------------|---|--------------------------------------|--|
| NAME | fn_ctx_equivalent_name – construct an equivalent name in same context | | |
| SYNOPSIS | <pre>#include <xfn/xfn.h> FN_composite_name_t *fn_ctx_equivalent_name(FN_ctx_t *ctx, const FN_composite_name_t *name, const FN_string_t *leading_name, FN_status_t *status);</pre> | | |
| DESCRIPTION | <p>Given the name of an object <i>name</i> relative to the context <i>ctx</i>, this operation returns an equivalent name for that object, relative to the same context <i>ctx</i>, that has <i>leading_name</i> as its initial atomic name. Two names are said to be equivalent if they have prefixes that resolve to the same context, and the parts of the names immediately following the prefixes are identical.</p> <p>The existence of a binding for <i>leading_name</i> in <i>ctx</i> does not guarantee that a name equivalent to <i>name</i> can be constructed. The failure may be because such equivalence is not meaningful, or due to the inability of the system to construct a name with the equivalence. For example, supplying <code>_thishost</code> as <i>leading_name</i> when <i>name</i> starts with <code>_myself</code> to <code>fn_ctx_equivalent_name()</code> in the Initial Context would not be meaningful; this results in the return of the error code <code>FN_E_NO_EQUIVALENT_NAME</code>.</p> | | |
| RETURN VALUES | If an equivalent name cannot be constructed, the value 0 is returned and <i>status</i> is set appropriately. | | |
| ERRORS | <p><code>fn_ctx_equivalent_name()</code> sets <i>status</i> as described in <code>FN_status_t(3N)</code> and <code>xfn_status_codes(3N)</code>. The following status code is especially relevant for this operation:</p> <table border="0" style="width: 100%;"> <tr> <td style="vertical-align: top; padding-right: 20px;"><code>FN_E_NO_EQUIVALENT_NAME</code></td> <td>No equivalent name can be constructed, either because there is no meaningful equivalence between <i>name</i> and <i>leading_name</i>, or the system does not support constructing the requested equivalent name, for implementation-specific reasons.</td> </tr> </table> | <code>FN_E_NO_EQUIVALENT_NAME</code> | No equivalent name can be constructed, either because there is no meaningful equivalence between <i>name</i> and <i>leading_name</i> , or the system does not support constructing the requested equivalent name, for implementation-specific reasons. |
| <code>FN_E_NO_EQUIVALENT_NAME</code> | No equivalent name can be constructed, either because there is no meaningful equivalence between <i>name</i> and <i>leading_name</i> , or the system does not support constructing the requested equivalent name, for implementation-specific reasons. | | |
| EXAMPLES | <p>EXAMPLE 1 Naming Files</p> <p>In the Initial Context supporting XFN enterprise policies, a user <code>jsmith</code> is able to name one of her files relative to this context in several ways.</p> <pre>_myself/_fs/map.ps _user/jsmith/_fs/map.ps _orgunit/finance/_user/jsmith/_fs/map.ps</pre> | | |

The first of these may be appealing to the user `jsmith` in her day-to-day operations. This name is not, however, appropriate for her to use when referring the file in an electronic mail message sent to a colleague. The second of these names would be appropriate if the colleague were in the same organizational unit, and the third appropriate for anyone in the same enterprise.

When the following sequence of instructions is executed by the user `jsmith` in the organizational unit `finance`, `enterprise_wide_name` would contain the composite name `_orgunit/finance/_user/jsmith/_fs/map.ps`:

```
FN_string_t* namestr =
    fn_string_from_str((const unsigned char*)_myself/_fs/map.ps");
FN_composite_name_t* name = fn_composite_name_from_string(namestr);
FN_string_t* org_lead =
    fn_string_from_str((const unsigned char*)_orgunit");
FN_status_t* status = fn_status_create();
FN_composite_name_t* enterprise_wide_name;
FN_ctx_t* init_ctx = fn_ctx_handle_from_initial(status);
/* check status of from_initial() */
enterprise_wide_name = fn_ctx_equivalent_name(init_ctx, name, org_lead,
status);
```

When the following sequence of instructions is executed by the user `jsmith` in the organizational unit `finance`, `shortest_name` would contain the composite name `_myself/_fs/map.ps`:

```
FN_string_t* namestr =
    fn_string_from_str((const unsigned char*)
        "_orgunit/finance/_user/jsmith/_fs/map.ps");
FN_composite_name_t* name = fn_composite_name_from_string(namestr);
FN_string_t* mylead = fn_string_from_str((const unsigned char*)_myself");
FN_status_t* status = fn_status_create();
FN_composite_name_t* shortest_name;
FN_ctx_t* init_ctx = fn_ctx_handle_from_initial(status);
/* check status of from_initial() */
shortest_name = fn_ctx_equivalent_name(init_ctx, name, mylead, status);
```

ATTRIBUTES

See **attributes** (5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

FN_composite_name_t(3N), **FN_ctx_t**(3N), **FN_status_t**(3N),
FN_string_t(3N), **xfn_status_codes**(3N), **attributes**(5)

| NAME | fn_ctx_get_ref – return a context’s reference | | | | |
|--------------------------|--|----------------|-----------------|----------|-------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lxfn [library ...] #include <xfn/xfn.h> FN_ref_t *fn_ctx_get_ref(const FN_ctx_t *ctx, FN_status_t *status);</pre> | | | | |
| DESCRIPTION | This operation returns a reference to the supplied context object. | | | | |
| RETURN VALUE | fn_ctx_get_ref() returns a pointer to an FN_ref_t object if the operation succeeds, it returns 0 if the operation fails. | | | | |
| ERRORS | <p>fn_ctx_get_ref() sets <i>status</i> as described in FN_status_t(3N) and xfn_status_codes(3N). The following status code is of particular relevance to this operation:</p> <p>FN_E_OPERATION_NOT_SUPPORTED Using the fn_ctx_get_ref() operation on the Initial Context returns this status code.</p> | | | | |
| APPLICATION USAGE | <p>fn_ctx_get_ref() cannot be used on the Initial Context. fn_ctx_get_ref() can be used on contexts bound in the Initial Context (in other words, the bindings in the Initial Context have references).</p> <p>If the context handle was created earlier using the fn_ctx_handle_from_ref() operation, the reference returned by the fn_ctx_get_ref() operation may not necessarily be exactly the same in content as that originally supplied. For example, fn_ctx_handle_from_ref() may construct the context handle from one address from the list of addresses. The context implementation may return with a call to fn_ctx_get_ref() only that address, or a more complete list of addresses than what was supplied in fn_ctx_handle_from_ref().</p> | | | | |
| ATTRIBUTES | See attributes (5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe.</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe. |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe. | | | | |
| SEE ALSO | FN_ctx_t(3N) , FN_ref_t(3N) , FN_status_t(3N) , fn_ctx_handle_from_initial(3N) , fn_ctx_handle_from_ref(3N) , xfn_status_codes (3N) , xfn(3N) , attributes(5) | | | | |

| NAME | fn_ctx_get_syntax_attrs – return syntax attributes associated with named context | | | | |
|--------------------------|---|----------------|-----------------|----------|-------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lxfn [library ...] #include <xfn/xfn.h></pre> <p>FN_attrset_t *fn_ctx_get_syntax_attrs(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);</p> | | | | |
| DESCRIPTION | <p>Each context has an associated set of syntax-related attributes. This operation returns the syntax attributes associated with the context named by <i>name</i> relative to the context <i>ctx</i>.</p> <p>The attributes must contain the attribute <code>fn_syntax_type</code> (<code>FN_ID_STRING</code> format). If the context supports a syntax that conforms to the XFN standard syntax model, <code>fn_syntax_type</code> is set to "standard" (ASCII attribute syntax) and the attribute set contains the rest of the relevant syntax attributes described in <code>xfn_compound_names(3N)</code>.</p> <p>This operation is different from other XFN attribute operations in that these syntax attributes could be obtained directly from the context. Attributes obtained through other XFN attribute operations may not necessarily be associated with the context; they may be associated with the reference of context, rather than the context itself (see <code>xfn_attributes(3N)</code>).</p> | | | | |
| RETURN VALUE | <code>fn_ctx_get_syntax_attrs()</code> returns an attribute set if successful; it returns a NULL pointer (0) if the operation fails. | | | | |
| ERRORS | <code>fn_ctx_get_syntax_attrs()</code> sets <i>status</i> as described in <code>FN_status_t(3N)</code> and <code>xfn_status_codes(3N)</code> . | | | | |
| APPLICATION USAGE | <p>Implementations may choose to support other syntax types in addition to, or in place of, the XFN standard syntax model, in which case, the value of the <code>fn_syntax_type</code> attribute would be set to an implementation-specific string, and different or additional syntax attributes will be in the set.</p> <p>Syntax attributes of a context may be generated automatically by a context, in response to <code>fn_ctx_get_syntax_attrs()</code>, or they may be created and updated using the base attribute operations. This is implementation-dependent.</p> | | | | |
| ATTRIBUTES | See <code>attributes</code> (5) for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe.</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe. |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe. | | | | |

SEE ALSO

FN_attrset_t(3N), FN_composite_name_t(3N),
FN_compound_name_t(3N), FN_ctx_t(3N), FN_status_t(3N),
fn_attr_get(3N), fn_attr_multi_get(3N),
xfn_compound_names(3N), xfn_attributes(3N),
xfn_status_codes(3N), xfn(3N), attributes(5)

NAME | fn_ctx_handle_destroy – release storage associated with context handle

SYNOPSIS | `cc [flag ...] file ... -lxfn [library ...]
#include <xfn/xfn.h>`

`void fn_ctx_handle_destroy(FN_ctx_t *ctx);`

DESCRIPTION | This operation destroys the context handle *ctx* and allows the implementation to free resources associated with the context handle. This operation does not affect the state of the context itself.

ATTRIBUTES | See **attributes** (5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe. |

SEE ALSO | **FN_ctx_t**(3N), **fn_ctx_handle_from_initial**(3N),
fn_ctx_handle_from_ref(3N), **xfn**(3N), **attributes**(5)

| | |
|----------------------|---|
| NAME | fn_ctx_handle_from_initial – return a handle to the Initial Context |
| SYNOPSIS | <pre>cc [flag ...] file ... -lxfn [library ...] #include <xfn/xfn.h></pre> <p>FN_ctx_t *fn_ctx_handle_from_initial(unsigned int <i>authoritative</i>, FN_status_t *<i>status</i>);</p> |
| DESCRIPTION | <p>This operation returns a handle to the caller's Initial Context. On successful return, the handle points to a context which meets the specification of the XFN Initial Context (see <code>fns_initial_context(5)</code>).</p> <p><i>authoritative</i> specifies whether the handle to the context returned should be authoritative with respect to information the context obtains from the naming service. When the flag is non-zero, subsequent operations on the context will access the most authoritative information. When <i>authoritative</i> is 0, the handle to the context returned need not be authoritative.</p> |
| RETURN VALUES | <code>fn_ctx_handle_from_initial()</code> returns a pointer to an <code>FN_ctx_t</code> object if the operation succeeds; it returns a <code>NULL</code> pointer (0) otherwise. |
| ERRORS | <code>fn_ctx_handle_from_initial()</code> sets only the status code portion of the status object <i>status</i> . |
| USAGE | <p>Authoritativeness is determined by specific naming services. For example, in a naming service that supports replication using a master/slave model, the source of authoritative information would come from the master server. In some naming systems, bypassing the naming service cache may reach servers which provide the most authoritative information. The availability of an authoritative context might be lower due to the lower number of servers offering this service. For the same reason, it might also provide poorer performance than contexts that need not be authoritative.</p> <p>Applications set <i>authoritative</i> to 0 for typical day-to-day operations. Applications only set <i>authoritative</i> to a non-zero value when they require access to the most authoritative information, possibly at the expense of lower availability and/or poorer performance.</p> <p>It is implementation-dependent whether authoritativeness is transferred from one context to the next as composite name resolution proceeds. Getting an authoritative context handle to the Initial Context means that operations on bindings in the Initial Context are processed using the most authoritative information. Contexts referenced implicitly through an authoritative Initial Context (for example, through the use of composite names) may not necessarily themselves be authoritative.</p> |

ATTRIBUTES

See **attributes** (5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

FN_ctx_t(3N), **FN_status_t**(3N), **fn_ctx_get_ref**(3N),
fn_ctx_handle_from_ref(3N), **xfn**(3N), **xfn_status_codes**(3N),
attributes(5), **fns_initial_context**(5)

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

| | | | |
|---------------------------|---|---------------------------|--|
| NAME | fn_ctx_handle_from_ref – construct a handle to a context object using the given reference | | |
| SYNOPSIS | <pre>cc [flag ...] file ... -lxfn [library ...] #include <xfn/xfn.h></pre> <p>FN_ctx_t *fn_ctx_handle_from_ref(const FN_ref_t *ref, unsigned int <i>authoritative</i>, FN_status_t *status);</p> | | |
| DESCRIPTION | <p>This operation creates a handle to an FN_ctx_t object using an FN_ref_t object for that context.</p> <p><i>authoritative</i> specifies whether the handle to the context returned should be authoritative with respect to information the context obtains from the naming service. When the flag is non-zero, subsequent operations on the context will access the most authoritative information. When <i>authoritative</i> is 0, the handle to the context returned need not be authoritative.</p> | | |
| RETURN VALUES | This operation returns a pointer to an FN_ctx_t object if the operation succeeds; otherwise, it returns a NULL pointer (0). | | |
| ERRORS | <p>fn_ctx_handle_from_ref() sets <i>status</i> as described in FN_status_t(3N) and xfn_status_codes(3N). The following status code is of particular relevance to this operation:</p> <table border="0" style="width: 100%;"> <tr> <td style="vertical-align: top; padding-right: 20px;">FN_E_NO_SUPPORTED_ADDRESS</td> <td>A context object could not be constructed from a particular reference. The reference contained no address type over which the context interface was supported.</td> </tr> </table> | FN_E_NO_SUPPORTED_ADDRESS | A context object could not be constructed from a particular reference. The reference contained no address type over which the context interface was supported. |
| FN_E_NO_SUPPORTED_ADDRESS | A context object could not be constructed from a particular reference. The reference contained no address type over which the context interface was supported. | | |
| USAGE | <p>Authoritativeness is determined by specific naming services. For example, in a naming service that supports replication using a master/slave model, the source of authoritative information would come from the master server. In some naming systems, bypassing the naming service cache may reach servers which provide the most authoritative information. The availability of an authoritative context might be lower due to the lower number of servers offering this service. For the same reason, it might also provide poorer performance than contexts that need not be authoritative.</p> <p>Applications set <i>authoritative</i> to 0 for typical day-to-day operations. Applications only set <i>authoritative</i> to a non-zero value when they require access to the most authoritative information, possibly at the expense of lower availability and/or poorer performance.</p> | | |

To control the authoritativeness of the target context, the application first resolves explicitly to the target context using `fn_ctx_lookup(3N)`. It then uses `fn_ctx_handle_from_ref()` with the appropriate authoritative argument to obtain a handle to the context. This returns a handle to a context with the specified authoritativeness. The application then uses the XFN operations, such as lookup and list, with this context handle.

It is implementation-dependent whether authoritativeness is transferred from one context to the next as composite name resolution proceeds. The application should use the approach recommended above to achieve the desired level of authoritativeness on a per context basis.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`FN_ctx_t(3N)`, `FN_ref_t(3N)`, `FN_status_t(3N)`, `fn_ctx_get_ref(3N)`, `fn_ctx_handle_destroy(3N)`, `fn_ctx_lookup(3N)`, `xfn(3N)`, `xfn_status_codes(3N)`, `attributes(5)`, `fns_references(5)`

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

NAME | fn_ctx_list_bindings, FN_bindinglist_t, fn_bindinglist_next, fn_bindinglist_destroy – list the atomic names and references bound in a context

SYNOPSIS | cc
 [*flag* ...]
file
 ...
 -lxfn
 [*library* ...]
 #include <xfn/xfn.h>

```
FN_bindinglist_t * fn_ctx_list_bindings(FN_ctx_t * ctx, const
FN_composite_name_t * name, FN_status_t * status);

FN_string_t * fn_bindinglist_next(FN_bindinglist_t * bl, FN_ref_t ** ref,
FN_status_t * status);

void fn_bindinglist_destroy(FN_bindinglist_t * bl, FN_status_t * status);
```

DESCRIPTION | This set of operations is used to list the names and bindings in the context named by *name* relative to the context *ctx* . Note that *name* must name a context. If the intent is to list the contents of *ctx* , *name* should be an empty composite name.

The semantics of these operations are similar to those for listing names (see **fn_ctx_list_names(3N)**). In addition to a name string being returned, **fn_bindinglist_next()** also returns the reference of the binding for each member of the enumeration.

ATTRIBUTES | See **attributes** (5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **FN_composite_name_t(3N)** , **FN_ctx_t(3N)** , **FN_ref_t(3N)** , **FN_status_t(3N)** , **FN_string_t(3N)** , **fn_ctx_list_names(3N)** , **xfn(3N)** , **xfn_status_codes(3N)** , **attributes(5)**

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

| | |
|----------------------|---|
| NAME | fn_ctx_list_names, FN_namelist_t, fn_namelist_next, fn_namelist_destroy – list the atomic names bound in a context |
| SYNOPSIS | <pre>cc [flag ...] file ... -lxfn [library ...] #include <xfn/xfn.h></pre> <pre>FN_namelist_t * fn_ctx_list_names(FN_ctx_t * ctx, const FN_composite_name_t * name, FN_status_t * status); FN_string_t * fn_namelist_next(FN_namelist_t * nl, FN_status_t * status); void fn_namelist_destroy(FN_namelist_t * nl, FN_status_t * status);</pre> |
| DESCRIPTION | <p>This set of operations is used to list the names bound in the target context named <i>name</i> relative to the context <i>ctx</i> . Note that <i>name</i> must name a context. If the intent is to list the contents of <i>ctx</i> , <i>name</i> should be an empty composite name.</p> <p>The call to fn_ctx_list_names() initiates the enumeration process. It returns a handle to an <code>FN_namelist_t</code> object that can be used to enumerate the names in the target context.</p> <p>The operation fn_namelist_next() returns the next name in the enumeration identified by <i>nl</i> and updates <i>nl</i> to indicate the state of the enumeration. Successive calls to fn_namelist_next() using <i>nl</i> return successive names in the enumeration and further update the state of the enumeration. fn_namelist_next() returns a <code>NULL</code> pointer (0) when the enumeration has been completed.</p> <p>fn_namelist_destroy() is used to release resources used during the enumeration. This may be invoked at any time to terminate the enumeration.</p> |
| RETURN VALUES | <p>fn_ctx_list_names() returns a pointer to an <code>FN_namelist_t</code> object if the enumeration is successfully initiated; otherwise it returns a <code>NULL</code> pointer (0).</p> <p>fn_namelist_next() returns a <code>NULL</code> pointer (0) if no more names can be returned in the enumeration.</p> |

In the case of a failure, these operations return in *status* a code indicating the nature of the failure.

ERRORS

Each successful call to **fn_namelist_next()** returns a name and sets *status* to `FN_SUCCESS`.

When **fn_namelist_next()** returns a `NULL` pointer (`0`), it indicates that no more names can be returned. *status* is set in the following way:

| | |
|---------------------------------------|---|
| <code>FN_SUCCESS</code> | The enumeration has completed successfully. |
| <code>FN_E_INVALID_ENUM_HANDLE</code> | The supplied enumeration handle is not valid. Possible reasons could be that the handle was from another enumeration, or the context being enumerated no longer accepts the handle (due to such events as handle expiration or updates to the context). |

| | |
|----------------------------------|--|
| <code>FN_E_PARTIAL_RESULT</code> | The enumeration is not yet complete but cannot be continued. |
|----------------------------------|--|

Other status codes, such as `FN_E_COMMUNICATION_FAILURE`, are also possible in calls to **fn_ctx_list_names()**, **fn_namelist_next()**, and **fn_namelist_destroy()**. These functions set *status* for these other status codes as described in **FN_status_t(3N)** and **xfn_status_codes(3N)**.

USAGE

The names enumerated using **fn_namelist_next()** are not ordered in any way. There is no guaranteed relation between the order in which names are added to a context and the order of names obtained by enumeration. The specification does *not* guarantee that any two series of enumerations will return the names in the same order.

When a name is added to or removed from a context, this may or may not invalidate the enumeration handle that the client holds for that context. If the enumeration handle becomes invalid, the status code

`FN_E_INVALID_ENUM_HANDLE` is returned in *status*. If the enumeration handle remains valid, the update may or may not be visible to the client.

In addition, there may be a relationship between the *ctx* argument supplied to **fn_ctx_list_names()** and the `FN_namelist_t` object it returns. For example, some implementations may store the context handle *ctx* within the `FN_namelist_t` object for subsequent **fn_namelist_next()** calls. In general, a **fn_ctx_handle_destroy(3N)** should not be invoked on *ctx* until the enumeration has terminated.

EXAMPLES**EXAMPLE 1** A sample program.

The following code fragment illustrates how the list names operations may be used:

```
extern FN_string_t *user_input;
FN_ctx_t *ctx;
FN_composite_name_t *target_name = fn_composite_name_from_string(user_input);
FN_status_t *status = fn_status_create();
FN_string_t *name;
FN_namelist_t *nl;
ctx = fn_ctx_handle_from_initial(status);
/* error checking on 'status' */
if ((nl=fn_ctx_list_names(ctx, target_name, status)) == 0) {
\011/* report 'status' and exit */
}
while (name=fn_namelist_next(nl, status)) {
\011/* do something with 'name' */
\011fn_string_destroy(name);
}
/* check 'status' for reason for end of enumeration and report if necessary */
/* clean up */
fn_namelist_destroy(nl, status);
/* report 'status' */
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

FN_composite_name_t(3N), **FN_ctx_t(3N)**, **FN_status_t(3N)**,
FN_string_t(3N), **fn_ctx_handle_destroy(3N)**, **xfn(3N)**,
xfn_status_codes(3N), **attributes(5)**

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

| NAME | fn_ctx_lookup – look up name in context | | | | |
|--------------------------|---|----------------|-----------------|----------|-------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lxfn [library ...] #include <xfn/xfn.h></pre> <p>FN_ref_t *fn_ctx_lookup(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);</p> | | | | |
| DESCRIPTION | This operation returns the reference bound to <i>name</i> relative to the context <i>ctx</i> . | | | | |
| RETURN VALUE | If the operation succeeds, the fn_ctx_lookup() function returns a handle to the reference bound to <i>name</i> . Otherwise, 0 is returned and <i>status</i> is set appropriately. | | | | |
| ERRORS | fn_ctx_lookup() sets <i>status</i> as described FN_status_t(3N) and xfn_status_codes(3N) . | | | | |
| APPLICATION USAGE | Some naming services may not always have reference information for all names in their contexts; for such names, such naming services may return a special reference whose type indicates that the name is not bound to any address. This reference may be name service specific or it may be the conventional NULL reference defined in the X/Open registry. See fns_references(5) . | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe.</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe. |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe. | | | | |
| SEE ALSO | FN_composite_name_t(3N) , FN_ctx_t(3N) , FN_ref_t(3N) , FN_status_t(3N) , fns_references(5) , xfn_status_codes(3N) , xfn(3N) , attributes(5) | | | | |

| NAME | fn_ctx_lookup_link – look up the link reference bound to a name | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lxfn [library ...] #include <xfn/xfn.h></pre> <p>FN_ref_t *fn_ctx_lookup_link(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);</p> | | | | |
| DESCRIPTION | <p>This operation returns the XFN link bound to <i>name</i>. The terminal atomic part of <i>name</i> must be bound to an XFN link.</p> <p>The normal <code>fn_ctx_lookup(3N)</code> operation follows all links encountered, including any bound to the terminal atomic part of <i>name</i>. This operation differs from the normal lookup in that when the terminal atomic part of <i>name</i> is an XFN link, this link is not followed, and the operation returns the link.</p> | | | | |
| RETURN VALUES | If <code>fn_ctx_lookup_link()</code> fails, a NULL pointer (0) is returned. | | | | |
| ERRORS | <p><code>fn_ctx_lookup_link()</code> sets <i>status</i> as described in <code>FN_status_t(3N)</code> and <code>xfn_status_codes(3N)</code>. Of special relevance for <code>fn_ctx_lookup_link()</code> is the following status code:</p> <p><code>FN_E_MALFORMED_LINK</code> <i>name</i> resolved to a reference that was not a link.</p> | | | | |
| ATTRIBUTES | See <code>attributes(5)</code> for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | <code>FN_composite_name_t(3N)</code> , <code>FN_ctx_t(3N)</code> , <code>FN_ref_t(3N)</code> , <code>FN_status_t(3N)</code> , <code>fn_ctx_lookup(3N)</code> , <code>xfn(3N)</code> , <code>xfn_links(3N)</code> , <code>xfn_status_codes(3N)</code> , <code>attributes(5)</code> | | | | |
| NOTES | The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification. | | | | |

NAME | fn_ctx_rename – rename the name of a binding

SYNOPSIS |

```
cc [ flag ... ] file ... -lxfn [ library ... ]
#include <xfn/xfn.h>
```

|

```
int fn_ctx_rename(FN_ctx_t *ctx, const FN_composite_name_t *oldname, const
FN_composite_name_t *newname, unsigned int exclusive, FN_status_t *status);
```

DESCRIPTION | The **fn_ctx_rename()** operation binds the reference currently bound to *oldname* relative to *ctx*, to the name *newname*, and unbinds *oldname*. *newname* is resolved relative to the target context (that named by all but the terminal atomic part of *oldname*).

| If *exclusive* is 0, the operation overwrites any old binding of *newname*. If *exclusive* is nonzero, the operation fails if *newname* is already bound.

RETURN VALUES | **fn_ctx_rename()** returns 1 if the operation is successful, 0 otherwise.

ERRORS | **fn_ctx_rename()** sets *status* as described **FN_status_t(3N)** and **xfn_status_codes(3N)**.

USAGE | The only restriction that XFN places on *newname* is that it be resolved relative to the target context. XFN does not specify further restrictions on *newname*. For example, in some implementations, *newname* might be restricted to be a name in the same naming system as the terminal component of *oldname*. In another implementation, *newname* might be restricted to be an atomic name.

| Normal resolution always follows links. In an **fn_ctx_rename()** operation, resolution of *oldname* continues to the target context; the terminal atomic name is not resolved. If the terminal atomic name is bound to a link, the link is not followed and the operation binds *newname* to the link and unbinds the terminal atomic name of *oldname*.

| In naming systems that support attributes and store the attributes along with the names, the unbind of the terminal atomic name of *oldname* also removes its associated attributes. It is implementation-dependent whether these attributes become associated with *newname*.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`FN_composite_name_t(3N)`, `FN_ctx_t(3N)`, `FN_ref_t(3N)`,
`FN_status_t(3N)`, `fn_ctx_bind(3N)` `fn_ctx_unbind(3N)`, `xfn(3N)`,
`xfn_status_codes(3N)`, `attributes(5)`

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

| | |
|-----------------|--|
| NAME | FN_ctx_t – an XFN context |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lxfn [<i>library</i> ...] #include <xfn/xfn.h> FN_ctx_t *fn_ctx_handle_from_initial(unsigned int <i>authoritative</i>, FN_status_t *<i>status</i>); FN_ctx_t *fn_ctx_handle_from_ref(const FN_ref_t *<i>ref</i>, unsigned int <i>authoritative</i>, FN_status_t *<i>status</i>); FN_ref_t *fn_ctx_get_ref(const FN_ctx_t *<i>ctx</i>, FN_status_t *<i>status</i>); void fn_ctx_handle_destroy(FN_ctx_t *<i>ctx</i>); FN_ref_t *fn_ctx_lookup(FN_ctx_t *<i>ctx</i>, const FN_composite_name_t *<i>name</i>, FN_status_t *<i>status</i>); FN_namelist_t *fn_ctx_list_names(FN_ctx_t *<i>ctx</i>, const FN_composite_name_t *<i>name</i>, FN_status_t *<i>status</i>); FN_string_t *fn_namelist_next(FN_namelist_t *<i>nl</i>, FN_status_t *<i>status</i>); void fn_namelist_destroy(FN_namelist_t *<i>nl</i>, FN_status_t *<i>status</i>); FN_bindinglist_t *fn_ctx_list_bindings(FN_ctx_t *<i>ctx</i>, const FN_composite_name_t *<i>name</i>, FN_status_t *<i>status</i>); FN_string_t *fn_bindinglist_next(FN_bindinglist_t *<i>iter</i>, FN_ref_t **<i>ref</i>, FN_status_t *<i>status</i>); void fn_bindinglist_destroy(FN_bindinglist_t *<i>iter_pos</i>, FN_status_t *<i>status</i>); int fn_ctx_bind(FN_ctx_t *<i>ctx</i>, const FN_composite_name_t *<i>name</i>, const FN_ref_t *<i>ref</i>, unsigned int <i>exclusive</i>, FN_status_t *<i>status</i>); int fn_ctx_unbind(FN_ctx_t *<i>ctx</i>, const FN_composite_name_t *<i>name</i>, FN_status_t *<i>status</i>); int fn_ctx_rename(FN_ctx_t *<i>ctx</i>, const FN_composite_name_t *<i>oldname</i>, const FN_composite_name_t *<i>newname</i>, unsigned int <i>exclusive</i>, FN_status_t *<i>status</i>); FN_ref_t *fn_ctx_create_subcontext(FN_ctx_t *<i>ctx</i>, const FN_composite_name_t *<i>name</i>, FN_status_t *<i>status</i>); int fn_ctx_destroy_subcontext(FN_ctx_t *<i>ctx</i>, const FN_composite_name_t *<i>name</i>, FN_status_t *<i>status</i>); FN_ref_t *fn_ctx_lookup_link(FN_ctx_t *<i>ctx</i>, const FN_composite_name_t *<i>name</i>, FN_status_t *<i>status</i>);</pre> |

DESCRIPTION

```
FN_attrset_t *fn_ctx_get_syntax_attrs(FN_ctx_t *ctx, const FN_composite_name_t
*name, FN_status_t *status);
```

An XFN context consists of a set of name to reference bindings. An XFN context is represented by the type `FN_ctx_t` in the client interface. The operations for manipulating an `FN_ctx_t` object are described in detail in separate reference manual pages.

The following contains a brief summary of these operations:

fn_ctx_handle_from_initial() returns a pointer to an Initial Context that provides a starting point for resolution of composite names.

fn_ctx_handle_from_ref() returns a handle to an `FN_ctx_t` object using the given reference *ref*. **fn_ctx_get_ref()** returns the reference of the context *ctx*.

fn_ctx_handle_destroy() releases the resources associated with the `FN_ctx_t` object *ctx*; it does not affect the state of the context itself.

fn_ctx_lookup() returns the reference bound to *name* resolved relative to *ctx*.

fn_ctx_list_names() is used to enumerate the atomic names bound in the context named by *name* resolved relative to *ctx*. **fn_ctx_list_bindings()** is used to enumerate the atomic names and their references in the context named by *name* resolved relative to *ctx*.

fn_ctx_bind() binds the composite name *name* to a reference *ref* resolved relative to *ctx*. **fn_ctx_unbind()** unbinds *name* resolved relative to *ctx*.

fn_ctx_rename() binds *newname* to the reference bound to *oldname* and unbinds *oldname*. *oldname* is resolved relative to *ctx*; *newname* is resolved relative to the target context.

fn_ctx_create_subcontext() creates a new context with the given composite name *name* resolved relative to *ctx*. **fn_ctx_destroy_subcontext()** destroys the context named by *name* resolved relative to *ctx*.

Normal resolution always follows links. **fn_ctx_lookup_link()** looks up *name* relative to *ctx*, following links except for the last atomic part of *name*, which must be bound to an XFN link.

fn_ctx_get_syntax_attrs() returns an attribute set containing attributes that describe a context's syntax. *name* must name a context.

ERRORS

In each context operation, the caller supplies an `FN_status_t` object as a parameter. The called function sets this status object as described in `FN_status_t(3N)` and `xfn_status_codes(3N)`.

USAGE

In most of the operations of the base context interface, the caller supplies a context and a composite name. The supplied name is always interpreted relative to the supplied context.

The operation may eventually be effected on a different context called the operation's *target context*. Each operation has an initial resolution phase that conveys the operation to its target context, and the operation is then applied. The effect (but not necessarily the implementation) is that of doing a lookup on that portion of the name that represents the target context, and then invoking the operation on the target context. The contexts involved only in the resolution phase are called *intermediate contexts*.

Normal resolution of names in context operations always follows XFN links.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`FN_attrset_t(3N)`, `FN_composite_name_t(3N)`, `FN_ref_t(3N)`,
`FN_status_t(3N)`, `fn_ctx_bind(3N)`, `fn_ctx_create_subcontext(3N)`,
`fn_ctx_destroy_subcontext(3N)`, `fn_ctx_get_ref(3N)`,
`fn_ctx_get_syntax_attrs(3N)`, `fn_ctx_handle_destroy(3N)`,
`fn_ctx_handle_from_initial(3N)`, `fn_ctx_handle_from_ref(3N)`,
`fn_ctx_list_bindings(3N)`, `fn_ctx_list_names(3N)`,
`fn_ctx_lookup(3N)`, `fn_ctx_lookup_link(3N)`, `fn_ctx_rename(3N)`,
`fn_ctx_unbind(3N)`, `xfn(3N)`, `xfn_links(3N)`, `xfn_status_codes(3N)`,
`attributes(5)`

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

| NAME | fn_ctx_unbind – unbind a name from a context | | | | |
|--------------------------|--|----------------|-----------------|----------|-------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lxfn [library ...] #include <xfn/xfn.h></pre> <pre>int fn_ctx_unbind(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status);</pre> | | | | |
| DESCRIPTION | <p>This operation removes the terminal atomic name in <i>name</i> from the the target context — that named by all but the terminal atomic part of <i>name</i>.</p> <p>This operation is successful even if the terminal atomic name was not bound in target context, but fails if any of the intermediate names are not bound. fn_ctx_unbind() is idempotent.</p> | | | | |
| RETURN VALUE | The operation returns 1 if successful, and 0 otherwise. | | | | |
| ERRORS | <p>fn_ctx_unbind() sets <i>status</i> as described in <code>FN_status_t</code> and <code>xfn_status_codes</code> (3N).</p> <p>Certain naming systems may disallow unbinding a name if the name is bound to an existing context in order to avoid orphan contexts that cannot be reached via any name. In such situations, the status code <code>FN_E_OPERATION_NOT_SUPPORTED</code> is returned.</p> | | | | |
| APPLICATION USAGE | <p>In naming systems that support attributes, and store the attributes along with the names, the unbind operation removes the name and its associated attributes.</p> <p>Normal resolution always follows links. In an fn_ctx_unbind() operation, resolution of <i>name</i> continues to the target context; the terminal atomic name is not resolved. If the terminal atomic name is bound to a link, the link is not followed and the link itself is unbound from the terminal atomic name.</p> | | | | |
| ATTRIBUTES | See <code>attributes(5)</code> for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe.</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe. |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe. | | | | |
| SEE ALSO | <code>FN_composite_name_t(3N)</code> , <code>FN_ctx_t(3N)</code> , <code>FN_ref_t(3N)</code> , <code>FN_status_t(3N)</code> , <code>fn_ctx_bind(3N)</code> , <code>fn_ctx_lookup(3N)</code> , <code>xfn_status_codes(3N)</code> , <code>xfn(3N)</code> , <code>attributes(5)</code> | | | | |

| | | | | | | | | | |
|----------------------|---|--------------|--|----------------|---|----------------------|--|-------------------|---|
| NAME | FN_identifier_t – an XFN identifier | | | | | | | | |
| DESCRIPTION | <p>Identifiers are used to identify reference types and address types in an XFN reference, and to identify attributes and their syntax in the attribute operations.</p> <p>An XFN identifier consists of an <code>unsigned int</code>, which determines the format of identifier, and the actual identifier, which is expressed as a sequence of octets.</p> <p>The representation of this structure is defined by XFN as follows:</p> <pre>typedef struct { unsigned int format; size_t length; void *contents; } FN_identifier_t;</pre> <p>XFN defines a small number of standard forms for identifiers:</p> <table border="0"> <tr> <td>FN_ID_STRING</td> <td>The identifier is an ASCII string (ISO 646).</td> </tr> <tr> <td>FN_ID_DCE_UUID</td> <td>The identifier is an OSF DCE UUID in string representation. (See the X/Open DCE RPC.)</td> </tr> <tr> <td>FN_ID_ISO_OID_STRING</td> <td>The identifier is an ISO OID in ASN.1 dot-separated integer list string format. (See the ISO ASN.1.)</td> </tr> <tr> <td>FN_ID_ISO_OID_BER</td> <td>The identifier is an ISO OID in ASN.1 Basic Encoding Rules (BER) format. (See the ISO BER.)</td> </tr> </table> | FN_ID_STRING | The identifier is an ASCII string (ISO 646). | FN_ID_DCE_UUID | The identifier is an OSF DCE UUID in string representation. (See the X/Open DCE RPC.) | FN_ID_ISO_OID_STRING | The identifier is an ISO OID in ASN.1 dot-separated integer list string format. (See the ISO ASN.1.) | FN_ID_ISO_OID_BER | The identifier is an ISO OID in ASN.1 Basic Encoding Rules (BER) format. (See the ISO BER.) |
| FN_ID_STRING | The identifier is an ASCII string (ISO 646). | | | | | | | | |
| FN_ID_DCE_UUID | The identifier is an OSF DCE UUID in string representation. (See the X/Open DCE RPC.) | | | | | | | | |
| FN_ID_ISO_OID_STRING | The identifier is an ISO OID in ASN.1 dot-separated integer list string format. (See the ISO ASN.1.) | | | | | | | | |
| FN_ID_ISO_OID_BER | The identifier is an ISO OID in ASN.1 Basic Encoding Rules (BER) format. (See the ISO BER.) | | | | | | | | |
| FILES | <code>#include <xfn/xfn.h></code> | | | | | | | | |
| SEE ALSO | <code>FN_attribute_t(3N)</code> , <code>FN_ref_addr_t(3N)</code> , <code>FN_ref_t(3N)</code> , <code>xfn(3N)</code> | | | | | | | | |
| NOTES | The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification. | | | | | | | | |

| | |
|----------------------|---|
| NAME | fnmatch – match filename or path name |
| SYNOPSIS | <pre>#include <fnmatch.h> int fnmatch(const char *pattern, const char *string, int flags);</pre> |
| DESCRIPTION | <p>The fnmatch() function matches patterns as described on the fnmatch(5) manual page. It checks the <i>string</i> argument to see if it matches the <i>pattern</i> argument.</p> <p>The <i>flags</i> argument modifies the interpretation of <i>pattern</i> and <i>string</i>. It is the bitwise inclusive OR of zero or more of the following flags defined in the header <code><fnmatch.h></code>.</p> <p>FNM_PATHNAME If set, a slash (/) character in <i>string</i> will be explicitly matched by a slash in <i>pattern</i>; it will not be matched by either the asterisk (*) or question-mark (?) special characters, nor by a bracket ([]) expression.</p> <p> If not set, the slash character is treated as an ordinary character.</p> <p>FNM_NOESCAPE If not set, a backslash character (\) in <i>pattern</i> followed by any other character will match that second character in <i>string</i>. In particular, “\\” will match a backslash in <i>string</i>.</p> <p> If set, a backslash character will be treated as an ordinary character.</p> <p>FNM_PERIOD If set, a leading period in <i>string</i> will match a period in <i>pattern</i>; where the location of “leading” is indicated by the value of FNM_PATHNAME:</p> <ul style="list-style-type: none"> ■ If FNM_PATHNAME is set, a period is “leading” if it is the first character in <i>string</i> or if it immediately follows a slash. ■ If FNM_PATHNAME is not set, a period is “leading” only if it is the first character of <i>string</i>. <p>If not set, no special restrictions are placed on matching a period.</p> |
| RETURN VALUES | If <i>string</i> matches the pattern specified by <i>pattern</i> , then fnmatch() returns 0. If there is no match, fnmatch() returns FNM_NOMATCH , which is defined in the |

header `<fnmatch.h>`. If an error occurs, **fnmatch()** returns another non-zero value.

USAGE

The **fnmatch()** function has two major uses. It could be used by an application or utility that needs to read a directory and apply a pattern against each entry. The **find(1)** utility is an example of this. It can also be used by the **pax(1)** utility to process its *pattern* operands, or by applications that need to match strings in a similar manner.

The name **fnmatch()** is intended to imply *filename* match, rather than *pathname* match. The default action of this function is to match filenames, rather than path names, since it gives no special significance to the slash character. With the `FNМ_PATHNAME` flag, **fnmatch()** does match path names, but without tilde expansion, parameter expansion, or special treatment for period at the beginning of a filename.

The **fnmatch()** function can be used safely in multithreaded applications, as long as **setlocale(3C)** is not being called to change the locale.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |
| CSI | Enabled |

SEE ALSO

find(1), **pax(1)**, **glob(3C)**, **setlocale(3C)**, **wordexp(3C)**, **attributes(5)**, **fnmatch(5)**

| | |
|--------------------|---|
| NAME | FN_ref_addr_t, fn_ref_addr_create, fn_ref_addr_destroy, fn_ref_addr_copy, fn_ref_addr_assign, fn_ref_addr_type, fn_ref_addr_length, fn_ref_addr_data, fn_ref_addr_description – an address in an XFN reference |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lxfn [<i>library</i> ...] #include <xfn/xfn.h> FN_ref_addr_t * fn_ref_addr_create(const FN_identifier_t * type, size_t length, const void * data); void fn_ref_addr_destroy(FN_ref_addr_t * addr); FN_ref_addr_t * fn_ref_addr_copy(const FN_ref_addr_t * addr); FN_ref_addr_t * fn_ref_addr_assign(FN_ref_addr_t * dst, const FN_ref_addr_t * src); const FN_identifier_t * fn_ref_addr_type(const FN_ref_addr_t * addr); size_t fn_ref_addr_length(const FN_ref_addr_t * addr); const void * fn_ref_addr_data(const FN_ref_addr_t * addr); FN_string_t * fn_ref_addr_description(const FN_ref_addr_t * addr, unsigned int detail, unsigned int * more_detail);</pre> |
| DESCRIPTION | <p>An XFN reference is represented by the type <code>FN_ref_t</code>. An object of this type contains a reference type and a list of addresses. Each address in the list is represented by an object of type <code>FN_ref_addr_t</code>. An address consists of an opaque data buffer and a type field, of type <code>FN_identifier_t</code>.</p> <p>fn_ref_addr_create() creates and returns an address with the given type and data. <i>length</i> indicates the size of the data. fn_ref_addr_destroy() releases the storage associated with the given address. fn_ref_addr_copy() returns a copy of the given address object. fn_ref_addr_assign() makes a copy of the address pointed to by <i>src</i> and assigns it to <i>dst</i>, releasing any old contents of <i>dst</i>. A pointer to the same object as <i>dst</i> is returned.</p> |

fn_ref_addr_type() returns the type of the given address.
fn_ref_addr_length() returns the size of the address in bytes.
fn_ref_addr_data() returns the contents of the address.

fn_ref_addr_description() returns the implementation-defined textual description of the address. It takes as arguments a number, *detail*, and a pointer to a number, *more_detail*. *detail* specifies the level of detail for which the description should be generated; the higher the number, the more detail is to be provided. If *more_detail* is 0, it is ignored. If *more_detail* is non-zero, it is set by the description operation to indicate the next level of detail available, beyond that specified by *detail*. If no higher level of detail is available, *more_detail* is set to *detail*.

USAGE

The address type of an `FN_ref_addr_t` object is intended to identify the mechanism that should be used to reach the object using that address. The client must interpret the contents of the opaque data buffer of the address based on the type of the address, and on the type of the reference that the address is in. However, this interpretation is intended to occur below the application layer. Most applications developers should not have to manipulate the contents of either address or reference objects themselves. These interfaces would generally be used within service libraries.

Multiple addresses in a single reference are intended to identify multiple communication endpoints for the same conceptual object. Multiple addresses may arise for various reasons, such as the object offering interfaces over more than one communication mechanism.

Manipulation of addresses using the operations described in this manual page does not affect their representation in the underlying naming system. Changes to addresses in the underlying naming system can only be effected through the use of the interfaces described in `FN_ctx_t(3N)`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`FN_ctx_t(3N)`, `FN_identifier_t(3N)`, `FN_ref_t(3N)`,
`FN_string_t(3N)`, `xfn(3N)`, `attributes(5)`

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications

developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

NAME FN_ref_t, fn_ref_create, fn_ref_destroy, fn_ref_copy, fn_ref_assign, fn_ref_type, fn_ref_addrcount, fn_ref_first, fn_ref_next, fn_ref_prepend_addr, fn_ref_append_addr, fn_ref_insert_addr, fn_ref_delete_addr, fn_ref_delete_all, fn_ref_create_link, fn_ref_is_link, fn_ref_link_name, fn_ref_description – an XFN reference

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lxfn
[
library
... ]
#include
e <xfn/xfn.h>

FN_ref_t * fn_ref_create(const FN_identifier_t * ref_type);

void fn_ref_destroy(FN_ref_t * ref);

FN_ref_t * fn_ref_copy(const FN_ref_t * ref);

FN_ref_t * fn_ref_assign(FN_ref_t * dst, const FN_ref_t * src);

const FN_identifier_t * fn_ref_type(const FN_ref_t * ref);

unsigned int fn_ref_addrcount(const FN_ref_t * ref);

const FN_ref_addr_t * fn_ref_first(const FN_ref_t * ref, void ** iter_pos);

const FN_ref_addr_t * fn_ref_next(const FN_ref_t * ref, void ** iter_pos);

int fn_ref_prepend_addr(FN_ref_t * ref, const FN_ref_addr_t * addr);

int fn_ref_append_addr(FN_ref_t * ref, const FN_ref_addr_t * addr);

int fn_ref_insert_addr(FN_ref_t * ref, void ** iter_pos, const FN_ref_addr_t * addr);

int fn_ref_delete_addr(FN_ref_t * ref, void ** iter_pos);

int fn_ref_delete_all(FN_ref_t * ref);

FN_ref_t * fn_ref_create_link(const FN_composite_name_t * link_name);

int fn_ref_is_link(const FN_ref_t * ref);

FN_composite_name_t * fn_ref_link_name(const FN_ref_t * link_ref);
```

DESCRIPTION

```
FN_string_t * fn_ref_description(const FN_ref_t * ref, unsigned int detail, unsigned
int * more_detail);
```

An XFN reference is represented by the type `FN_ref_t`. An object of this type contains a reference type and a list of addresses. The ordering in this list at the time of binding might not be preserved when the reference is returned upon lookup.

The reference type is represented by an object of type `FN_identifier_t`. The reference type is intended to identify the class of object referenced. XFN does not dictate the precise use of this.

Each address is represented by an object of type `FN_ref_addr_t`.

fn_ref_create() creates a reference with no address, using *ref_type* as its reference type. Addresses can be added later to the reference using the functions described below. **fn_ref_destroy()** releases the storage associated with *ref*. **fn_ref_copy()** creates a copy of *ref* and returns it. **fn_ref_assign()** creates a copy of *src* and assigns it to *dst*, releasing any old contents of *dst*. A pointer to the same object as *dst* is returned.

fn_ref_addrcount() returns the number of addresses in the reference *ref*.

fn_ref_first() returns the first address in *ref* and sets *iter_pos* to be after the address. It returns 0 if there is no address in the list. **fn_ref_next()** returns the address following *iter_pos* in *ref* and sets *iter_pos* to be after the address. If the iteration marker *iter_pos* is at the end of the sequence, **fn_ref_next()** returns 0.

fn_ref_prepend_addr() adds *addr* to the front of the list of addresses in *ref*.

fn_ref_append_addr() adds *addr* to the end of the list of addresses in *ref*.

fn_ref_insert_addr() adds *addr* to *ref* before *iter_pos* and sets *iter_pos* to be immediately after the new reference added. **fn_ref_delete_addr()** deletes the address located before *iter_pos* in the list of addresses in *ref* and sets *iter_pos* back one address. **fn_ref_delete_all ()** deletes all addresses in *ref*.

fn_ref_create_link() creates a reference using the given composite name *link_name* as an address. **fn_ref_is_link()** tests if *ref* is a link. It returns 1 if it is; 0 if it is not. **fn_ref_link_name()** returns the composite name stored in a link reference. It returns 0 if *link_ref* is not a link.

fn_ref_description() returns a string description of the given reference. It takes as argument an integer, *detail*, and a pointer to an integer, *more_detail*. *detail* specifies the level of detail for which the description should be generated; the higher the number, the more detail is to be provided. If *more_detail* is 0, it is ignored. If *more_detail* is non-zero, it is set by the description operation to indicate the next level of detail available, beyond that specified by *detail*. If no higher level of detail is available, *more_detail* is set to *detail*.

RETURN VALUES

The following operations return 1 if the operation succeeds, 0 if the operation fails:

fn_ref_prepend_addr()

fn_ref_append_addr()

fn_ref_insert_addr()

fn_ref_delete_addr()

fn_ref_delete_all()

USAGE

The reference type is intended to identify the class of object referenced. XFN does not dictate the precise use of this.

Multiple addresses in a single reference are intended to identify multiple communication endpoints for the same conceptual object. Multiple addresses may arise for various reasons, such as the object offering interfaces over more than one communication mechanism.

The client must interpret the contents of a reference based on the type of the addresses and the type of the reference. However, this interpretation is intended to occur below the application layer. Most applications developers should not have to manipulate the contents of either address or reference objects themselves. These interfaces would generally be used within service libraries.

Manipulation of references using the operations described in this manual page does not affect their representation in the underlying naming system. Changes to references in the underlying naming system can only be effected through the use of the interfaces described in **FN_ctx_t(3N)**.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

FN_composite_name_t(3N), **FN_ctx_t(3N)**, **FN_identifier_t(3N)**, **FN_ref_addr_t(3N)**, **FN_string_t(3N)**, **fn_ctx_lookup(3N)**, **fn_ctx_lookup_link(3N)**, **xfn(3N)**, **xfn_links(3N)**, **attributes(5)**

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next

minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

| | | | | | |
|------------------------|---|------------------------|-------------------------------------|-----------------------|--------------------------------|
| NAME | FN_search_control_t, fn_search_control_create, fn_search_control_destroy, fn_search_control_copy, fn_search_control_assign, fn_search_control_scope, fn_search_control_follow_links, fn_search_control_max_names, fn_search_control_return_ref, fn_search_control_return_attr_ids – options for attribute search | | | | |
| SYNOPSIS | <pre>#include <xfn/xfn.h> FN_search_control_t * fn_search_control_create(unsigned int scope, unsigned int follow_links, unsigned int max_names, unsigned int return_ref, const FN_attrset_t * return_attr_ids, unsigned int * status); void fn_search_control_destroy(FN_search_control_t * scontrol); FN_search_control_t * fn_search_control_copy(const FN_search_control_t * scontrol); FN_search_control_t * fn_search_control_assign(FN_search_control_t * dst, const FN_search_control_t * src); unsigned int fn_search_control_scope(const FN_search_control_t * scontrol); unsigned int fn_search_control_follow_links(const FN_search_control_t * scontrol); unsigned int fn_search_control_max_names(const FN_search_control_t * scontrol); unsigned int fn_search_control_return_ref(const FN_search_control_t * scontrol); const FN_attrset_t * fn_search_control_return_attr_ids(const FN_search_control_t * scontrol);</pre> | | | | |
| DESCRIPTION | <p>The FN_search_control_t object is used to specify options for the attribute search operation fn_attr_ext_search(3N) .</p> <p>fn_search_control_create() creates an FN_search_control_t object using information in <i>scope</i> , <i>follow_links</i> , <i>max_names</i> , <i>return_ref</i> , and <i>return_attr_ids</i> to set the search options. If the operation succeeds, fn_search_control_create() returns a pointer to an FN_search_control_t object; otherwise, it returns a NULL pointer.</p> <p>The scope of the search, <i>scope</i> , is either the named object, the named context, the named context and its subcontexts, or the named context and a context implementation defined set of subcontexts. The values for <i>scope</i> are:</p> <table border="0"> <tr> <td style="padding-right: 20px;">FN_SEARCH_NAMED_OBJECT</td> <td>Search just the given named object.</td> </tr> <tr> <td>FN_SEARCH_ONE_CONTEXT</td> <td>Search just the given context.</td> </tr> </table> | FN_SEARCH_NAMED_OBJECT | Search just the given named object. | FN_SEARCH_ONE_CONTEXT | Search just the given context. |
| FN_SEARCH_NAMED_OBJECT | Search just the given named object. | | | | |
| FN_SEARCH_ONE_CONTEXT | Search just the given context. | | | | |

FN_SEARCH_SUBTREE Search given context and all its subcontexts.

FN_SEARCH_CONSTRAINED_SUBTREE Search given context and its subcontexts as constrained by the context-specific policy in place at the named context.

follow_links further defines the scope and nature of the search. If *follow_links* is nonzero, the search follows XFN links. If *follow_links* is 0, XFN links are not followed. See `fn_attr_ext_search(3N)` for more detail about how XFN links are treated.

max_names specifies the maximum number of names to return in an `FN_ext_searchlist_t(3N)` enumeration (see `fn_attr_ext_search(3N)`). The names of all objects whose attributes satisfy the filter are returned when *max_names* is 0.

If *return_ref* is non-zero, the reference bound to the named object is returned with the object's name by `fn_ext_searchlist_next(3N)` (see `fn_attr_ext_search(3N)`). If *return_ref* is 0, the reference is not returned.

Attribute identifiers and values associated with named objects that satisfy the filter may be returned by `fn_ext_searchlist_next(3N)`. The attributes returned are those listed in *return_attr_ids*. If the value of *return_attr_ids* is 0, all attributes are returned. If *return_attr_ids* is an empty `FN_attrset_t` object (see `FN_attrset_t(3N)`), no attributes are returned. Any attribute values in *return_attr_ids* are ignored; only the attribute identifiers are relevant for this operation.

`fn_attr_ext_search(3N)` interprets a value of 0 for the search control argument as a default search control which has the following option settings:

| | |
|------------------------|--|
| scope | FN_SEARCH_ONE_CONTEXT |
| follow_links | 0 (do not follow links) |
| max_names | 0 (return all named objects that match filter) |
| return_ref | 0 (do not return the reference of the named object) |
| return_attr_ids | an empty <code>FN_attrset_t</code> object (do not return any attributes of the named object) |

`fn_search_control_destroy()` releases the storage associated with *scontrol*.

`fn_search_control_copy()` returns a copy of the search control *scontrol*.

fn_search_control_assign() makes a copy of the search control *src* and assigns it to *dst* , releasing the old contents of *dst* . A pointer to the same object as *dst* is returned.

fn_search_control_scope() returns the scope for the search.

fn_search_control_follow_links() returns non-zero if links are followed; 0 if not.

fn_search_control_max_names() returns the maximum number of names.

fn_search_control_return_ref() returns nonzero if the reference is returned; 0 if not.

fn_search_control_return_attr_ids() returns a pointer to the list of attributes; a NULL pointer indicates that all attributes and values are returned.

ERRORS

fn_search_control_create() returns a NULL pointer if the operation fails and sets status as follows:

FN_E_SEARCH_INVALID_OPTION A supplied search option was invalid or inconsistent.

Other status codes are possible (see **xfn_status_codes(3N)**).

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

FN_attrset_t(3N) , **fn_attr_ext_search(3N)** ,
xfn_status_codes(3N) , **attributes(5)**

| | |
|--------------------|--|
| NAME | FN_search_filter_t, fn_search_filter_create, fn_search_filter_destroy, fn_search_filter_copy, fn_search_filter_assign, fn_search_filter_expression, fn_search_filter_arguments – filter expression for attribute search |
| SYNOPSIS | <pre>#include <xfn/xfn.h> FN_search_filter_t * fn_search_filter_create(unsigned int * status, const unsigned char * estr, .); void fn_search_filter_destroy(FN_search_filter_t * sfilter); FN_search_filter_t * fn_search_filter_copy(const FN_search_filter_t * sfilter); FN_search_filter_t * fn_search_filter_assign(FN_search_filter_t * dst, const FN_search_filter_t * src); const char * fn_search_filter_expression(const FN_search_filter_t * sfilter); const void ** fn_search_filter_arguments(const FN_search_filter_t * sfilter, size_t * number_of_arguments);</pre> |
| DESCRIPTION | <p>The FN_search_filter_t type is an expression that is evaluated against the attributes of named objects bound in the scope of the search operation fn_attr_ext_search(3N). The filter evaluates to TRUE or FALSE. If the filter is empty, it evaluates to TRUE. Names of objects whose attribute values satisfy the filter expression are returned by the search operation.</p> <p>If the identifier in any subexpression of the filter does not exist as an attribute of an object, then the innermost logical expression containing that identifier is FALSE. A subexpression that is only an attribute tests for the presence of the attribute; the subexpression evaluates to TRUE if the attribute has been defined for the object and FALSE otherwise.</p> <p>fn_search_filter_create() creates a search filter from the expression string <i>estr</i> and the remaining arguments.</p> <p>fn_search_filter_destroy() releases the storage associated with the search filter <i>sfilter</i>.</p> <p>fn_search_filter_copy() returns a copy of the search filter <i>sfilter</i>.</p> <p>fn_search_filter_assign() makes a copy of the search filter <i>src</i> and assigns it to <i>dst</i>, releasing the old contents of <i>dst</i>. A pointer to the same object as <i>dst</i> is returned.</p> <p>fn_search_filter_expression() returns the filter expression of <i>sfilter</i>.</p> <p>fn_search_filter_arguments() returns an array of pointers to arguments supplied to the filter constructor. <i>number_of_arguments</i> is set to the size of this</p> |

array. The types of the arguments are determined by the substitution tokens in the expression in *sfilter*.

BNF of Filter Expression

```

<FilterExpr> ::= [ <Expr> ]
<Expr> ::= <Expr> "or" <Expr>
\011      <Expr> "and" <Expr>
\011      | "not" <Expr>
\011      | "(" <Expr> ")"
          | <Attribute> [ <Rel_Op> <Value> ]
\011      | <Ext>
<Rel_Op> ::= "=" | "!=" | "<" | "<=" | ">" | ">=" | "~="
<Attribute> ::= "%a"
<Value> ::= <Integer>
\011      | "%v"
\011      | <Wildcarded_string>
<Wildcarded_string> ::= "*"
\011      | <String>
          | {<String> "*" }+ [ <String> ]
          | { "*" <String> }+
          | [ "*" ]

<String>
::= "\"" {
<Char>
} * "\""
\011      | "%s"

<Char>
::=
<PCS> // See BNF in Section 4.1.2 for PCSdefinition
\011      | Characters in the repertoire of a string representation
<Identifier> ::= "%i"
<Ext> ::= <Ext_Op> "(" [Arg_List] ")"
<Ext_Op> ::= <String> | <Identifier>
<Arg_List> ::= <Arg> | <Arg> "," <Arg_List>
<Arg> ::= <Value> | <Attribute> | <Identifier>

```

Specification of Filter Expression

The arguments to **fn_search_filter_create()** are a return status, an expression string, and a list of arguments. The string contains the filter expression with substitution tokens for the attributes, attribute values, strings, and identifiers that are part of the expression. The remaining list of arguments contains the attributes and values in the order of appearance of their corresponding substitution tokens in the expression. The arguments are of types `FN_attribute_t*`, `FN_attrvalue_t*`, `FN_string_t*`, or `FN_identifier_t*`. Any attribute values in an `FN_attribute_t*` type of argument are ignored; only the attribute identifier and attribute syntax are relevant. The argument type expected by each substitution token are listed in the following table.

| Token | Argument Type |
|-------|------------------|
| %a | FN_attribute_t* |
| %v | FN_attrvalue_t* |
| %s | FN_string_t* |
| %i | FN_identifier_t* |

Precedence

The following precedence relations hold in the absence of parentheses, in the order of lowest to highest:

- or
- and
- not
- relational operators

These boolean and relational operators are left associative.

Relational Operators

Comparisons and ordering are specific to the syntax and/or rules of the supplied attribute.

Locale (code set, language, or territory) mismatches that occur during string comparisons and ordering operations are resolved in an implementation-dependent way. Relational operations that have ordering semantics may be used for strings of code sets in which ordering is meaningful, but is not of general use in internationalized environments.

An attribute that occurs in the absence of any relational operator tests for the presence of the attribute.

| Operator | Meaning |
|----------|---|
| == | The sub-expression is TRUE if at least one value of the specified attribute is equal to the supplied value. |
| != | The sub-expression is TRUE if no values of the specified attribute equal the supplied value. |
| >= | The sub-expression is TRUE if at least one value of the attribute is greater than or equal to the supplied value. |
| > | The sub-expression is TRUE if at least one value of the attribute is greater then the supplied value. |

| Operator | Meaning |
|----------|--|
| < = | The sub-expression is TRUE if at least one value of the attribute is less than or equal to the supplied value. |
| < | The sub-expression is TRUE if at least one value of the attribute is less than the supplied value. |
| ≈ = | The sub-expression is TRUE if at least one value of the specified attribute matches the supplied value according to some context-specific approximate matching criterion. This criterion must subsume strict equality. |

Wildcarded Strings

A wildcarded string consists of a sequence of alternating wildcard specifiers and strings. The sequence can start with either a wildcard specifier or a string, and end with either a wildcard specifier or a string.

The wildcard specifier is denoted by the asterisk character (' * ') and means zero or more occurrences of any character.

Wildcarded strings can be used to specify substring matches. The following are examples of wildcarded strings and what they mean:

| Wildcarded String | Meaning |
|--|--|
| * | Any string |
| *'ing' | Any string ending with ing |
| Any string starting with jo , and containing the substring ph , and which contains the substring ne in the portion of the string following ph , and which ends with er | |
| T} | |
| %s* | Any string starting with the supplied string |

| Wildcarded String | Meaning |
|--|---------|
| Any string starting with <code>bix</code> and ending with the supplied string <code>T}</code> | |

String matches involving strings of different locales (code set, language, or territory) are resolved in an implementation-dependent way.

Extended Operations

In addition to the relational operators, extended operators can be specified. All extended operators return either `TRUE` or `FALSE`. A filter expression can contain both relational and extended operations.

Extended operators are specified using an identifier (see `FN_identifier_t(3N)`) or a string. If the operator is specified using a string, the string is used to construct an identifier of format `FN_ID_STRING`. Identifiers of extended operators and signatures of the corresponding extended operations, as well as their suggested semantics, are registered with X/Open Company Ltd.

The following three extended operations are currently defined:

- 'name' (< *Wildcarded String*) The identifier for this operation is 'name' (`FN_ID_STRING`). The argument to this operation is a wildcard string. The operation returns `TRUE` if the name of the object matches the supplied wildcard string.

- 'reftype' (%i) The identifier for this operation is 'reftype' (`FN_ID_STRING`). The argument to this operation is an identifier. The operation returns `TRUE` if the reference type of the object is equal to the supplied identifier.

- 'addrtype' (%i) The identifier for this operation is 'addrtype' (`LM FN_ID_STRING`). The argument to the operation is an identifier. The operation returns `TRUE` if any of the address types in the reference of the object is equal to the supplied identifier.

Support and exact semantics of extended operations are context-specific. If a context does not support an extended operation, or if the filter expression

supplies the extended operation with either an incorrect number or type of arguments, the error `FN_E_SEARCH_INVALID_OP` is returned. (Note: `FN_E_OPERATION_NOT_SUPPORTED` is returned when `fn_attr_ext_search(3N)` is not supported.)

The following are examples of filter expressions that contain extended operations:

| Expression | Meaning |
|--|---|
| Evaluates to TRUE if the name of the object starts with bill. T} | |
| <code>%i(%a, %v)</code> | Evaluates to result of applying the specified operation to the supplied arguments. |
| <code>(%a == %v) and 'name'('joe'*)</code> | Evaluates to TRUE if the specified attribute has the given value and if the name of the object starts with <code>joe</code> . |

RETURN VALUES

`fn_search_filter_create()` returns a pointer to an `FN_search_filter_t` object if the operation succeeds; otherwise it returns a NULL pointer.

ERRORS

`fn_search_filter_create()` returns a NULL pointer if the operation fails and sets *status* in the following way:

- `FN_E_SEARCH_INVALID_FILTER` The filter expression had a syntax error or some other problem.
- `FN_E_SEARCH_INVALID_OP` An operator in the filter expression is not supported or, if the operator is an extended operator, the number of types of arguments supplied does not match the signature of the operation.
- `FN_E_INVALID_ATTR_IDENTIFIER` The left hand side of an operator expression was not an attribute.

FN_E_INVALID_ATTR_VALUE The right hand side of an operator expression was not an integer, attribute value, or (wildcarded) string.

Other status codes are possible as described in the reference manual pages for `FN_status_t(3N)` and `xfn_status_codes(3N)`.

EXAMPLES

EXAMPLE 1 Creating Different Filters

The following examples illustrate how to create three different filters.

The first example shows how to construct a filter involving substitution tokens and literals in the same filter expression. This example creates a filter for named objects whose `color` attribute contains a string value of `red`, `blue`, or `white`. The first two values are specified using substitution tokens; the last value, `white`, is specified as a literal in the expression.

```
unsigned int status;
extern FN_attribute_t *attr_color;
FN_string_t *red = fn_string_from_str((unsigned char *)"red");
FN_string_t *blue = fn_string_from_str((unsigned char *)"blue");
FN_search_filter_t *sfilter;
sfilter = fn_search_filter_create(
    \011&status,
    \011"(%a == %s) or (%a == %s) or (%a == 'white')",
    \011attr_color, red, attr_color, blue,
    \011attr_color);
```

The second example illustrates how to construct a filter involving a wildcarded string. This example creates a filter for searching for named objects whose `last_name` attribute has a value that begins with the character `m`.

```
unsigned int status;
extern FN_attribute_t *attr_last_name;
FN_search_filter_t *sfilter;
sfilter = fn_search_filter_create(
    \011&status, "%a == 'm'", attr_last_name);
```

The third example illustrates how to construct a filter involving extended operations. This example creates a filter for finding all named objects whose name ends with `ton`.

```
unsigned int status;
FN_search_filter_t *sfilter;
sfilter = fn_search_filter_create(&status, "'name'(*'ton')");
```

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`FN_attribute_t(3N)`, `FN_attrvalue_t(3N)`, `FN_identifier_t(3N)`,
`FN_status_t(3N)`, `FN_string_t(3N)`, `fn_attr_ext_search(3N)`,
`xfn_status_codes(3N)`, `attributes(5)`

NAME FN_status_t, fn_status_create, fn_status_destroy, fn_status_copy, fn_status_assign, fn_status_code, fn_status_remaining_name, fn_status_resolved_name, fn_status_resolved_ref, fn_status_diagnostic_message, fn_status_link_code, fn_status_link_remaining_name, fn_status_link_resolved_name, fn_status_link_resolved_ref, fn_status_link_diagnostic_message, fn_status_is_success, fn_status_set_success, fn_status_set, fn_status_set_code, fn_status_set_remaining_name, fn_status_set_resolved_name, fn_status_set_resolved_ref, fn_status_set_diagnostic_message, fn_status_set_link_code, fn_status_set_link_remaining_name, fn_status_set_link_resolved_name, fn_status_set_link_resolved_ref, fn_status_set_link_diagnostic_message, fn_status_append_resolved_name, fn_status_append_remaining_name, fn_status_advance_by_name, fn_status_description – an XFN status object

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lxfn
[
library
... ]
#include <xfn/xfn.h>

FN_status_t * fn_status_create(void);

void fn_status_destroy(FN_status_t * stat);

FN_status_t * fn_status_copy(const FN_status_t * stat);

FN_status_t * fn_status_assign(FN_status_t * dst, const FN_status_t * src);

unsigned int fn_status_code(const FN_status_t * stat);

const FN_composite_name_t * fn_status_remaining_name(const FN_status_t * stat);

const FN_composite_name_t * fn_status_resolved_name(const FN_status_t * stat);

const FN_ref_t * fn_status_resolved_ref(const FN_status_t * stat);

const FN_string_t * fn_status_diagnostic_message(const FN_status_t * stat);

unsigned int fn_status_link_code(const FN_status_t * stat);

const FN_composite_name_t * fn_status_link_remaining_name(const FN_status_t
* stat);
```

```
const FN_composite_name_t * fn_status_link_resolved_name(const FN_status_t *
stat);

const FN_ref_t * fn_status_link_resolved_ref(const FN_status_t * stat);

const FN_string_t * fn_status_link_diagnostic_message(const FN_status_t * stat);

int fn_status_is_success(const FN_status_t * stat);

int fn_status_set_success(FN_status_t * stat);

int fn_status_set(FN_status_t * stat, unsigned int code, const FN_ref_t * resolved_ref,
const FN_composite_name_t * resolved_name, const FN_composite_name_t *
remaining_name);

int fn_status_set_code(FN_status_t * stat, unsigned int code);

int fn_status_set_remaining_name(FN_status_t * stat, const
FN_composite_name_t * name);

int fn_status_set_resolved_name(FN_status_t * stat, const FN_composite_name_t
* name);

int fn_status_set_resolved_ref(FN_status_t * stat, const FN_ref_t * ref);

int fn_status_set_diagnostic_message(FN_status_t * stat, const FN_string_t *
msg);

int fn_status_set_link_code(FN_status_t * stat, unsigned int code);

int fn_status_set_link_remaining_name(FN_status_t * stat, const
FN_composite_name_t * name);

int fn_status_set_link_resolved_name(FN_status_t * stat, const
FN_composite_name_t * name);

int fn_status_set_link_resolved_ref(FN_status_t * stat, const FN_ref_t * ref);

int fn_status_set_link_diagnostic_message(FN_status_t * stat, const
FN_string_t * msg);

int fn_status_append_resolved_name(FN_status_t * stat, const
FN_composite_name_t * name);

int fn_status_append_remaining_name(FN_status_t * stat, const
FN_composite_name_t * name);

int fn_status_advance_by_name(FN_status_t * stat, const FN_composite_name_t *
prefix, const FN_ref_t * resolved_ref);

FN_string_t * fn_status_description(const FN_status_t * stat, unsigned int detail,
unsigned int * more_detail);
```

DESCRIPTION

The result status of operations in the context interface and the attribute interface is encapsulated in an `FN_status_t` object. This object contains information about how the operation completed: whether an error occurred in performing the operation, the nature of the error, and information that helps locate where the error occurred. In the case that the error occurred while resolving an XFN link, the status object contains additional information about that error.

The context status object consists of several items of information:

| | |
|--------------------------------|---|
| primary status code | An unsigned int code describing the disposition of the operation. |
| resolved name | In the case of a failure during the resolution phase of the operation, this is the leading portion of the name that was resolved successfully. Resolution may have been successful beyond this point, but the error might not be pinpointed further. |
| resolved reference | The reference to which resolution was successful (in other words, the reference to which the resolved name is bound). |
| remaining name | The remaining unresolved portion of the name. |
| diagnostic message | This contains any diagnostic message returned by the context implementation. This message provides the context implementation a way of notifying the end-user or administrator of any implementation-specific information related to the returned error status. The diagnostic message could then be used by the end-user or administrator to take appropriate out-of-band action to rectify the problem. |
| link status code | In the case that an error occurred while resolving an XFN link, the primary status code has the value <code>FN_E_LINK_ERROR</code> and the link status code describes the error that occurred while resolving the XFN link. |
| resolved link name | In the case of a link error, this contains the resolved portion of the name in the XFN link. |
| resolved link reference | In the case of a link error, this contains the reference to which the resolved link name is bound. |

remaining link name In the case of a link error, this contains the remaining unresolved portion of the name in the XFN link.

link diagnostic message In the case of a link error, this contains any diagnostic message related to the resolution of the link.

Both the primary status code and the link status code are values of type `unsigned int` that are drawn from the same set of meaningful values. XFN reserves the values 0 through 127 for standard meanings. The values and interpretations for the codes are determined by XFN. See `xfn_status_codes(3N)`.

`fn_status_create()` creates a status object with status `FN_SUCCESS`. `fn_status_destroy()` releases the storage associated with `stat`. `fn_status_copy()` returns a copy of the status object `stat`. `fn_status_assign()` makes a copy of the status object `src` and assigns it to `dst`, releasing any old contents of `dst`. A pointer to the same object as `dst` is returned.

`fn_status_code()` returns the status code. `fn_status_remaining_name()` returns the remaining part of name to be resolved. `fn_status_resolved_name()` returns the part of the composite name that has been resolved.

`fn_status_resolved_ref()` returns the reference to which resolution was successful. `fn_status_diagnostic_message` returns any diagnostic message set by the context implementation.

`fn_status_link_code()` returns the link status code.

`fn_status_link_remaining_name()` returns the remaining part of the link name that has not been resolved. `fn_status_link_resolved_name()` returns the part of the link name that has been resolved. `fn_status_link_resolved_ref()` returns the reference to which resolution of the link was successful.

`fn_status_link_diagnostic_message()` returns any diagnostic message set by the context implementation during resolution of the link.

`fn_status_is_success()` returns 1 if the status indicates success, 0 otherwise.

`fn_status_set_success()` sets the status code to `FN_SUCCESS` and clears all other parts of `stat`. `fn_status_set()` sets the non-link contents of the status object `stat`. `fn_status_set_code()` sets the primary status code field of the status object `stat`. `fn_status_set_remaining_name()` sets the remaining name part of the status object `stat` to `name`. `fn_status_set_resolved_name()` sets the resolved name part of the status object `stat` to `name`. `fn_status_set_resolved_ref()` sets the resolved reference part of the status object `stat` to `ref`.

`fn_status_set_diagnostic_message()` sets the diagnostic message part of the status object to `msg`.

fn_status_set_link_code() sets the link status code field of the status object *stat* to indicate why resolution of the link failed.

fn_status_set_link_remaining_name() sets the remaining link name part of the status object *stat* to *name* . **fn_status_set_link_resolved_name()** sets the resolved link name part of the status object *stat* to *name* .

fn_status_set_link_resolved_ref() sets the resolved link reference part of the status object *stat* to *ref* . **fn_status_set_link_diagnostic_message()** sets the link diagnostic message part of the status object to *msg* .

fn_status_append_resolved_name() appends as additional components *name* to the resolved name part of the status object *stat* .

fn_status_append_remaining_name() appends as additional components *name* to the remaining name part of the status object *stat* .

fn_status_advance_by_name() removes *prefix* from the remaining name, and appends it to the resolved name. The resolved reference part is set to *resolved_ref* . This operation returns 1 on success, 0 if the *prefix* is not a prefix of the remaining name.

RETURN VALUES

The **fn_status_set_***() operations return 1 if the operation succeeds, 0 if the operation fails.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

FN_composite_name_t(3N) , **FN_ref_t(3N)** , **FN_string_t(3N)** , **xfn(3N)** , **xfn_status_codes(3N)** , **attributes(5)**

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

NAME FN_string_t, fn_string_create, fn_string_destroy, fn_string_from_str, fn_string_from_str_n, fn_string_str, fn_string_from_contents, fn_string_code_set, fn_string_charcount, fn_string_bytecount, fn_string_contents, fn_string_copy, fn_string_assign, fn_string_from_strings, fn_string_from_substring, fn_string_is_empty, fn_string_compare, fn_string_compare_substring, fn_string_next_substring, fn_string_prev_substring - a character string

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lxfn
[
library
... ]
#include <xfn/xfn.h>

FN_string_t * fn_string_create(void);

void fn_string_destroy(FN_string_t * str);

FN_string_t * fn_string_from_str(const unsigned char * cstr);
FN_string_t * fn_string_from_str_n(const unsigned char * cstr, size_t n);
const unsigned char * fn_string_str(const FN_string_t * str, unsigned int * status);
FN_string_t * fn_string_from_contents(unsigned long code_set, const void *
locale_info, size_t locale_info_len, size_t charcount, size_t bytecount, const void * contents,
unsigned int * status);
unsigned long fn_string_code_set(const FN_string_t * str, const void ** locale_info,
size_t * locale_info_len);
size_t fn_string_charcount(const FN_string_t * str);
size_t fn_string_bytecount(const FN_string_t * str);
const void * fn_string_contents(const FN_string_t * str);
FN_string_t * fn_string_copy(const FN_string_t * str);
FN_string_t * fn_string_assign(FN_string_t * dst, const FN_string_t * src);
FN_string_t * fn_string_from_strings(unsigned int * status, const FN_string_t * s1,
const FN_string_t * s2, ...);
```

```

FN_string_t * fn_string_from_substring(const FN_string_t * str, int first, int last);
int fn_string_is_empty(const FN_string_t * str);
int fn_string_compare(const FN_string_t * str1, const FN_string_t * str2, unsigned
int string_case, unsigned int * status);
int fn_string_compare_substring(const FN_string_t * str1, int first, int last, const
FN_string_t * str2, unsigned int string_case, unsigned int * status);
int fn_string_next_substring(const FN_string_t * str, const FN_string_t * sub, int
index, unsigned int string_case, unsigned int * status);
int fn_string_prev_substring(const FN_string_t * str, const FN_string_t * sub, int
index, unsigned int string_case, unsigned int * status);

```

DESCRIPTION

The `FN_string_t` type is used to represent character strings in the XFN interface. It provides insulation from specific string representations.

The `FN_string_t` supports multiple code sets. It provides creation functions for character strings of the code set of the current locale setting and a generic creation function for arbitrary code sets. The degree of support for the functions that manipulate `FN_string_t` for arbitrary code sets is implementation-dependent. An XFN implementation is required to support the ISO 646 code set; all other code sets are optional.

fn_string_destroy() releases the storage associated with the given string.

fn_string_create() creates an empty string.

fn_string_from_str() creates an `FN_string_t` object from the given null terminated string based on the code set of the current locale setting. The number of characters in the string is determined by the code set of the current locale setting. **fn_string_from_str_n()** is like **fn_string_from_str()** except only *n* characters from the given string are used. **fn_string_str()** returns the contents of the given string *str* in the form of a null terminated string in the code set and current locale setting.

fn_string_from_contents() creates an `FN_string_t` object using the specified code set *code_set*, locale information *locale_info*, and data in the given buffer *contents*. *bytecount* specifies the number of bytes in *contents* and *charcount* specifies the number of characters represented by *contents*.

fn_string_code_set() returns the code set associated with the given string object and, if present, the locale information in *locale_info*.

fn_string_charcount() returns the number of characters in the given string object. **fn_string_bytecount()** returns the number of bytes used to represent the given string object. **fn_string_contents()** returns a pointer to the contents of the given string object.

fn_string_copy() returns a copy of the given string object. **fn_string_assign()** makes a copy of the string object *src* and assigns it to *dst*, releasing any old contents of *dst*. A pointer to the same object as *dst* is returned.

fn_string_from_strings() is a function that takes a variable number of arguments (minimum of 2), the last of which must be `NULL (0)`; it returns a new string object composed of the left to right concatenation of the given strings, in the given order. The support for strings with different code sets and/or locales as arguments to a single invocation of **fn_string_from_strings()** is implementation-dependent. **fn_string_from_substring()** returns a new string object consisting of the characters located between *first* and *last* inclusive from *str*. Indexing begins with 0. If *last* is `FN_STRING_INDEX_LAST` or exceeds the length of the string, the index of the last character of the string is used.

fn_string_is_empty() returns whether *str* is an empty string.

Comparison of two strings must take into account code set and locale information. If strings are in the same code set and same locale, case sensitivity is applied according to the case sensitivity rules applicable for the code set and locale; case sensitivity may not necessarily be relevant for all string encodings. If *string_case* is non-zero, case is significant and equality for strings of the same code set is defined as equality between byte-wise encoded values of the strings. If *string_case* is zero, case is ignored and equality for strings of the same code set is defined using the definition of case-insensitive equality for the specific code set. Support for comparison between strings of different code sets, or lack thereof, is implementation-dependent.

fn_string_compare() compares strings *str1* and *str2* and returns 0 if they are equal, non-zero if they are not equal. If two strings are not equal, **fn_string_compare()** returns a positive value if the difference of *str2* precedes that of *str1* in terms of byte-wise encoded value (with case-sensitivity taken into account when *string_case* is non-zero), and a negative value if the difference of *str1* precedes that of *str2*, in terms of byte-wise encoded value (with case-sensitivity taken into account when *string_case* is non-zero). Such information (positive versus negative return value) may be used by applications that use strings of code sets in which ordering is meaningful; this information is not of general use in internationalized environments.

fn_string_compare_substring() is similar to **fn_string_compare()** except that **fn_string_compare_substring()** compares characters between *first* and *last* inclusive of *str2* with *str1*. Comparison of strings with incompatible code sets returns a negative or positive value (never 0) depending on the implementation.

fn_string_next_substring() returns the index of the next occurrence of *sub* at or after *index* in the string *str*. `FN_STRING_INDEX_NONE` is returned if *sub* does not occur. **fn_string_prev_substring()** returns the index of the previous occurrence of *sub* at or before *index* in the string *str*. `FN_STRING_INDEX_NONE`

is returned if *sub* does not occur. In both of these functions, *string_case* specifies whether the search should take case-sensitivity into account.

ERRORS

fn_string_str() returns 0 and sets *status* to `FN_E_INCOMPATIBLE_CODE_SETS` if the given string's representation cannot be converted into the code set of the current locale setting. It is implementation-dependent which code sets can be converted into the code set of the current locale.

Code set mismatches that occur during concatenation, searches, or comparisons are resolved in an implementation-dependent way. When an implementation discovers that arguments to substring searches and comparison operations have incompatible code sets, it sets *status* to `FN_E_INCOMPATIBLE_CODE_SETS`. In such cases, **fn_string_from_strings()** returns 0. The returned value for comparison operations when there is code set or locale incompatibility is either negative or positive (greater than 0); it is never 0.

fn_string_from_contents() returns 0 and *status* is set to `FN_E_INCOMPATIBLE_CODE_SETS` if the supplied code set and/or locale information are not supported by the XFN implementation.

ATTRIBUTES

See **attributes** (5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

xfn(3N) , **attributes(5)**

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

| | |
|--------------------|--|
| NAME | fopen, freopen – open a stream |
| SYNOPSIS | <pre> /usr/ucb/cc [<i>flag</i> ...] <i>file</i> ... #include <stdio.h> FILE * fopen(<i>file</i>, <i>mode</i>); const char * <i>file</i> , * <i>mode</i> ; FILE * freopen(<i>file</i>, <i>mode</i>, <i>iop</i>); const char *<i>file</i> , * <i>mode</i> ; register FILE * <i>iop</i> ; </pre> |
| DESCRIPTION | <p>fopen() opens the file named by <i>file</i> and associates a stream with it. If the open succeeds, fopen() returns a pointer to be used to identify the stream in subsequent operations.</p> <p><i>file</i> points to a character string that contains the name of the file to be opened.</p> <p><i>mode</i> is a character string having one of the following values:</p> <ul style="list-style-type: none"> r open for reading w truncate or create for writing a append: open for writing at end of file, or create for writing r+ open for update (reading and writing) w+ truncate or create for update a+ append; open or create for update at EOF |

freopen() opens the file named by *file* and associates the stream pointed to by *iop* with it. The *mode* argument is used just as in **fopen()**. The original stream is closed, regardless of whether the open ultimately succeeds. If the open succeeds, **freopen()** returns the original value of *iop*.

freopen() is typically used to attach the preopened streams associated with *stdin*, *stdout*, and *stderr* to other files.

When a file is opened for update, both input and output may be done on the resulting stream. However, output may not be directly followed by input without an intervening **fseek(3S)** or **rewind(3S)**, and input may not be directly followed by output without an intervening **fseek(3S)** or **rewind(3S)**. An input operation which encounters EOF will fail.

RETURN VALUES

fopen() and **freopen()** return a `NULL` pointer on failure.

SEE ALSO

open(2), **fclose(3S)**, **fopen(3S)**, **freopen(3S)**, **fseek(3S)**, **malloc(3C)**, **rewind(3S)**

NOTES

Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

In order to support the same number of open files that the system does, **fopen()** must allocate additional memory for data structures using **malloc(3C)** after 64 files have been opened. This confuses some programs which use their own memory allocators.

The interfaces of **fopen()** and **freopen()** differ from the Standard I/O Functions **fopen(3S)** and **freopen(3S)**. The Standard I/O Functions distinguish binary from text files with an additional use of 'b' as part of the *mode*. This enables portability of **fopen(3S)** and **freopen(3S)** beyond SunOS 4. X systems.

| | | | | | | | | | | | | | |
|--------------------|--|---------|------------------------|---------|---|---------|---|------------------|---|------------------|--|------------------|---|
| NAME | fopen – open a stream | | | | | | | | | | | | |
| SYNOPSIS | <pre>#include <stdio.h> FILE *fopen(const char *filename, const char *mode);</pre> | | | | | | | | | | | | |
| DESCRIPTION | <p>The fopen() function opens the file whose pathname is the string pointed to by <i>filename</i>, and associates a stream with it.</p> <p>The argument <i>mode</i> points to a string beginning with one of the following sequences:</p> <table border="0" style="margin-left: 20px;"> <tr> <td style="padding-right: 20px;">r or rb</td> <td>Open file for reading.</td> </tr> <tr> <td>w or wb</td> <td>Truncate to zero length or create file for writing.</td> </tr> <tr> <td>a or ab</td> <td>Append; open or create file for writing at end-of-file.</td> </tr> <tr> <td>r+ or rb+ or r+b</td> <td>Open file for update (reading and writing).</td> </tr> <tr> <td>w+ or wb+ or w+b</td> <td>Truncate to zero length or create file for update.</td> </tr> <tr> <td>a+ or ab+ or a+b</td> <td>Append; open or create file for update, writing at end-of-file.</td> </tr> </table> <p>The character <i>b</i> has no effect, but is allowed for ISO C standard conformance (see standards(5)). Opening a file with read mode (<i>r</i> as the first character in the <i>mode</i> argument) fails if the file does not exist or cannot be read.</p> <p>Opening a file with append mode (<i>a</i> as the first character in the <i>mode</i> argument) causes all subsequent writes to the file to be forced to the then current end-of-file, regardless of intervening calls to fseek(3S). If two separate processes open the same file for append, each process may write freely to the file without fear of destroying output being written by the other. The output from the two processes will be intermixed in the file in the order in which it is written.</p> <p>When a file is opened with update mode (<i>+</i> as the second or third character in the <i>mode</i> argument), both input and output may be performed on the associated stream. However, output must not be directly followed by input without an intervening call to fflush(3S) or to a file positioning function (fseek(3S), fsetpos(3S) or rewind(3S)), and input must not be directly followed by output without an intervening call to a file positioning function, unless the input operation encounters end-of-file.</p> <p>When opened, a stream is fully buffered if and only if it can be determined not to refer to an interactive device. The error and end-of-file indicators for the stream are cleared.</p> | r or rb | Open file for reading. | w or wb | Truncate to zero length or create file for writing. | a or ab | Append; open or create file for writing at end-of-file. | r+ or rb+ or r+b | Open file for update (reading and writing). | w+ or wb+ or w+b | Truncate to zero length or create file for update. | a+ or ab+ or a+b | Append; open or create file for update, writing at end-of-file. |
| r or rb | Open file for reading. | | | | | | | | | | | | |
| w or wb | Truncate to zero length or create file for writing. | | | | | | | | | | | | |
| a or ab | Append; open or create file for writing at end-of-file. | | | | | | | | | | | | |
| r+ or rb+ or r+b | Open file for update (reading and writing). | | | | | | | | | | | | |
| w+ or wb+ or w+b | Truncate to zero length or create file for update. | | | | | | | | | | | | |
| a+ or ab+ or a+b | Append; open or create file for update, writing at end-of-file. | | | | | | | | | | | | |

If *mode* is *w*, *a*, *w+* or *a+* and the file did not previously exist, upon successful completion, **fopen()** function will mark for update the *st_atime*, *st_ctime* and *st_mtime* fields of the file and the *st_ctime* and *st_mtime* fields of the parent directory.

If *mode* is *w* or *w+* and the file did previously exist, upon successful completion, **fopen()** will mark for update the *st_ctime* and *st_mtime* fields of the file. The **fopen()** function will allocate a file descriptor as **open(2)** does.

The largest value that can be represented correctly in an object of type *off_t* will be established as the offset maximum in the open file description.

RETURN VALUES

Upon successful completion, **fopen()** returns a pointer to the object controlling the stream. Otherwise, a null pointer is returned and *errno* is set to indicate the error.

The **fopen()** function may fail and not set *errno* if there are no free *stdio* streams.

ERRORS

The **fopen()** function will fail if:

| | |
|---------------------|--|
| EACCES | Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by <i>mode</i> are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created. |
| EINTR | A signal was caught during the execution of fopen() . |
| EISDIR | The named file is a directory and <i>mode</i> requires write access. |
| ELOOP | Too many symbolic links were encountered in resolving <i>path</i> . |
| EMFILE | There are <i>OPEN_MAX</i> file descriptors currently open in the calling process. |
| ENAMETOOLONG | The length of the <i>filename</i> exceeds <i>PATH_MAX</i> or a pathname component is longer than <i>NAME_MAX</i> . |
| ENFILE | The maximum allowable number of files is currently open in the system. |
| ENOENT | A component of <i>filename</i> does not name an existing file or <i>filename</i> is an empty string. |

| | |
|--|---|
| ENOSPC | The directory or file system that would contain the new file cannot be expanded, the file does not exist, and it was to be created. |
| ENOTDIR | A component of the path prefix is not a directory. |
| ENXIO | The named file is a character special or block special file, and the device associated with this special file does not exist. |
| EOVERFLOW | The current value of the file position cannot be represented correctly in an object of type <code>fpos_t</code> . |
| EROFS | The named file resides on a read-only file system and <i>mode</i> requires write access. |
| The <code>fopen()</code> function may fail if: | |
| EINVAL | The value of the <i>mode</i> argument is not valid. |
| EMFILE | The number of streams currently open in the calling process is either <code>FOPEN_MAX</code> or <code>STREAM_MAX</code> . |
| ENAMETOOLONG | Pathname resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code> . |
| ENOMEM | Insufficient storage space is available. |
| ETXTBSY | The file is a pure procedure (shared text) file that is being executed and <i>mode</i> requires write access. |

USAGE

The number of streams that a process can have open at one time is `STREAM_MAX`. If defined, it has the same value as `FOPEN_MAX`.

The `fopen()` function has a transitional interface for 64-bit file offsets. See `1f64(5)`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`fclose(3S)`, `fdopen(3S)`, `fflush(3S)`, `freopen(3S)`, `fsetpos(3S)`, `rewind(3S)`, `attributes(5)`, `1f64(5)`, `standards(5)`

| NAME | form_cursor, pos_form_cursor – position forms window cursor | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lform -lcurses [<i>library</i> ..] #include <form.h> int pos_form_cursor(FORM * <i>form</i>);</pre> | | | | |
| DESCRIPTION | pos_form_cursor() moves the form window cursor to the location required by the form driver to resume form processing. This may be needed after the application calls a <code>curses</code> library I/O routine. | | | | |
| RETURN VALUES | pos_form_cursor() returns one of the following: <p>E_OK The function returned successfully.</p> <p>E_SYSTEM_ERROR System error.</p> <p>E_BAD_ARGUMENT An argument is incorrect.</p> <p>E_NOT_POSTED The form is not posted.</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | curses(3X) , forms(3X) , attributes(5) | | | | |
| NOTES | The header <code><form.h></code> automatically includes the headers <code><eti.h></code> and <code><curses.h></code> . | | | | |

NAME form_data, data Ahead, data_behind – tell if forms field has off-screen data ahead or behind

SYNOPSIS

```
cc
[
  flag
  ... ]
file
...
-lform

-lcurses
[
  library
  .. ]
#include <form.h>

int data Ahead(FORM * form);

int data_behind(FORM * form);
```

DESCRIPTION **data Ahead()** returns TRUE (1) if the current field has more off-screen data ahead; otherwise it returns FALSE (0).
data_behind() returns TRUE (1) if the current field has more off-screen data behind; otherwise it returns FALSE (0).

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO **curses(3X)**, **forms(3X)**, **attributes(5)**

NOTES The header <form.h> automatically includes the headers <eti.h> and <curses.h> .

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------|---|----------------------|------------------------|----------------------|----------------------------|-----------------------|-------------------------|----------------------|------------------------|-----------------------|-------------------------|-----------------------|-----------------------------|------------------------|--------------------------|-----------------------|-------------------------|------------------------|--------------------------------|------------------------|--------------------------------|-------------------------|---------------------------------|------------------------|--------------------------------|-----------------------|---------------------|------------------------|----------------------|---------------------|-------------------|-----------------------|---------------------|----------------------|--|
| NAME | form_driver - command processor for the forms subsystem | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SYNOPSIS | <pre>cc [flag ...] file ... -lform -lcurses [library ..] #include <form.h></pre> <pre>int form_driver(FORM *form, int c);</pre> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| DESCRIPTION | <p>form_driver() is the workhorse of the <code>forms</code> subsystem; it checks to determine whether the character <code>c</code> is a <code>forms</code> request or data. If it is a request, the form driver executes the request and reports the result. If it is data (a printable ASCII character), it enters the data into the current position in the current field. If it is not recognized, the form driver assumes it is an application-defined command and returns <code>E_UNKNOWN_COMMAND</code>. Application defined commands should be defined relative to <code>MAX_COMMAND</code>, the maximum value of a request listed below.</p> <p>Form driver requests:</p> <table border="0"> <tr> <td>REQ_NEXT_PAGE</td> <td>Move to the next page.</td> </tr> <tr> <td>REQ_PREV_PAGE</td> <td>Move to the previous page.</td> </tr> <tr> <td>REQ_FIRST_PAGE</td> <td>Move to the first page.</td> </tr> <tr> <td>REQ_LAST_PAGE</td> <td>Move to the last page.</td> </tr> <tr> <td>REQ_NEXT_FIELD</td> <td>Move to the next field.</td> </tr> <tr> <td>REQ_PREV_FIELD</td> <td>Move to the previous field.</td> </tr> <tr> <td>REQ_FIRST_FIELD</td> <td>Move to the first field.</td> </tr> <tr> <td>REQ_LAST_FIELD</td> <td>Move to the last field.</td> </tr> <tr> <td>REQ_SNEXT_FIELD</td> <td>Move to the sorted next field.</td> </tr> <tr> <td>REQ_SPREV_FIELD</td> <td>Move to the sorted prev field.</td> </tr> <tr> <td>REQ_SFIRST_FIELD</td> <td>Move to the sorted first field.</td> </tr> <tr> <td>REQ_SLAST_FIELD</td> <td>Move to the sorted last field.</td> </tr> <tr> <td>REQ_LEFT_FIELD</td> <td>Move left to field.</td> </tr> <tr> <td>REQ_RIGHT_FIELD</td> <td>Move right to field.</td> </tr> <tr> <td>REQ_UP_FIELD</td> <td>Move up to field.</td> </tr> <tr> <td>REQ_DOWN_FIELD</td> <td>Move down to field.</td> </tr> <tr> <td>REQ_NEXT_CHAR</td> <td>Move to the next character in the field.</td> </tr> </table> | REQ_NEXT_PAGE | Move to the next page. | REQ_PREV_PAGE | Move to the previous page. | REQ_FIRST_PAGE | Move to the first page. | REQ_LAST_PAGE | Move to the last page. | REQ_NEXT_FIELD | Move to the next field. | REQ_PREV_FIELD | Move to the previous field. | REQ_FIRST_FIELD | Move to the first field. | REQ_LAST_FIELD | Move to the last field. | REQ_SNEXT_FIELD | Move to the sorted next field. | REQ_SPREV_FIELD | Move to the sorted prev field. | REQ_SFIRST_FIELD | Move to the sorted first field. | REQ_SLAST_FIELD | Move to the sorted last field. | REQ_LEFT_FIELD | Move left to field. | REQ_RIGHT_FIELD | Move right to field. | REQ_UP_FIELD | Move up to field. | REQ_DOWN_FIELD | Move down to field. | REQ_NEXT_CHAR | Move to the next character in the field. |
| REQ_NEXT_PAGE | Move to the next page. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| REQ_PREV_PAGE | Move to the previous page. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| REQ_FIRST_PAGE | Move to the first page. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| REQ_LAST_PAGE | Move to the last page. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| REQ_NEXT_FIELD | Move to the next field. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| REQ_PREV_FIELD | Move to the previous field. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| REQ_FIRST_FIELD | Move to the first field. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| REQ_LAST_FIELD | Move to the last field. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| REQ_SNEXT_FIELD | Move to the sorted next field. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| REQ_SPREV_FIELD | Move to the sorted prev field. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| REQ_SFIRST_FIELD | Move to the sorted first field. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| REQ_SLAST_FIELD | Move to the sorted last field. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| REQ_LEFT_FIELD | Move left to field. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| REQ_RIGHT_FIELD | Move right to field. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| REQ_UP_FIELD | Move up to field. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| REQ_DOWN_FIELD | Move down to field. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| REQ_NEXT_CHAR | Move to the next character in the field. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| | |
|-----------------------|--|
| REQ_PREV_CHAR | Move to the previous character in the field. |
| REQ_NEXT_LINE | Move to the next line in the field. |
| REQ_PREV_LINE | Move to the previous line in the field. |
| REQ_NEXT_WORD | Move to the next word in the field. |
| REQ_PREV_WORD | Move to the previous word in the field. |
| REQ_BEG_FIELD | Move to the first char in the field. |
| REQ_END_FIELD | Move after the last char in the field. |
| REQ_BEG_LINE | Move to the beginning of the line. |
| REQ_END_LINE | Move after the last char in the line. |
| REQ_LEFT_CHAR | Move left in the field. |
| REQ_RIGHT_CHAR | Move right in the field. |
| REQ_UP_CHAR | Move up in the field. |
| REQ_DOWN_CHAR | Move down in the field. |
| REQ_NEW_LINE | Insert/overlay a new line. |
| REQ_INS_CHAR | Insert the blank character at the cursor. |
| REQ_INS_LINE | Insert a blank line at the cursor. |
| REQ_DEL_CHAR | Delete the character at the cursor. |
| REQ_DEL_PREV | Delete the character before the cursor. |
| REQ_DEL_LINE | Delete the line at the cursor. |
| REQ_DEL_WORD | Delete the word at the cursor. |
| REQ_CLR_EOL | Clear to the end of the line. |
| REQ_CLR_EOF | Clear to the end of the field. |
| REQ_CLR_FIELD | Clear the entire field. |
| REQ_OVL_MODE | Enter overlay mode. |
| REQ_INS_MODE | Enter insert mode. |
| REQ_SCR_FLINE | Scroll the field forward a line. |
| REQ_SCR_BLINE | Scroll the field backward a line. |
| REQ_SCR_FPAGE | Scroll the field forward a page. |

| | |
|------------------------|---|
| REQ_SCR_BPAGE | Scroll the field backward a page. |
| REQ_SCR_FHPAGE | Scroll the field forward half a page. |
| REQ_SCR_BHPAGE | Scroll the field backward half a page. |
| REQ_SCR_FCHAR | Horizontal scroll forward a character. |
| REQ_SCR_BCHAR | Horizontal scroll backward a character. |
| REQ_SCR_HFLINE | Horizontal scroll forward a line. |
| REQ_SCR_HBLINE | Horizontal scroll backward a line. |
| REQ_SCR_HFHALL | Horizontal scroll forward half a line. |
| REQ_SCR_HBHALL | Horizontal scroll backward half a line. |
| REQ_VALIDATION | Validate field. |
| REQ_PREV_CHOICE | Display the previous field choice. |
| REQ_NEXT_CHOICE | Display the next field choice. |

RETURN VALUES

form_driver() returns one of the following:

| | |
|--------------------------|--|
| E_OK | The function returned successfully. |
| E_SYSTEM_ERROR | System error. |
| E_BAD_ARGUMENT | An argument is incorrect. |
| E_NOT_POSTED | The form is not posted. |
| E_INVALID_FIELD | The field contents are invalid. |
| E_BAD_STATE | The routine was called from an initialization or termination function. |
| E_REQUEST_DENIED | The form driver request failed. |
| E_UNKNOWN_COMMAND | An unknown request was passed to the form driver. |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO | `curses(3X)`, `forms(3X)`, `attributes(5)`

NOTES | The header `<form.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

| | | | | | | | | | |
|-----------------------|---|-------------|-------------------------------------|--------------------|---|-----------------------|---------------|-----------------------|--------------------------|
| NAME | form_field, set_form_fields, form_fields, field_count, move_field – connect fields to forms | | | | | | | | |
| SYNOPSIS | <pre>cc [flag ...] file ... -lform -lcurses [library ..] #include <form.h> int set_form_fields(FORM * form, FIELD ** field); FIELD ** form_fields(FORM * form); int field_count(FORM * form); int move_field(FIELD * field, int frow, int fcol);</pre> | | | | | | | | |
| DESCRIPTION | <p>set_form_fields() changes the fields connected to <i>form</i> to <i>fields</i> . The original fields are disconnected.</p> <p>form_fields() returns a pointer to the field pointer array connected to <i>form</i> .</p> <p>field_count() returns the number of fields connected to <i>form</i> .</p> <p>move_field() moves the disconnected <i>field</i> to the location <i>frow</i>, <i>fcol</i> in the forms subwindow.</p> | | | | | | | | |
| RETURN VALUES | <p>form_fields() returns NULL on error.</p> <p>field_count() returns -1 on error.</p> <p>set_form_fields() and move_field() return one of the following:</p> <table border="0"> <tr> <td style="padding-right: 20px;">E_OK</td> <td>The function returned successfully.</td> </tr> <tr> <td>E_CONNECTED</td> <td>The field is already connected to a form.</td> </tr> <tr> <td>E_SYSTEM_ERROR</td> <td>System error.</td> </tr> <tr> <td>E_BAD_ARGUMENT</td> <td>An argument is incorrect</td> </tr> </table> | E_OK | The function returned successfully. | E_CONNECTED | The field is already connected to a form. | E_SYSTEM_ERROR | System error. | E_BAD_ARGUMENT | An argument is incorrect |
| E_OK | The function returned successfully. | | | | | | | | |
| E_CONNECTED | The field is already connected to a form. | | | | | | | | |
| E_SYSTEM_ERROR | System error. | | | | | | | | |
| E_BAD_ARGUMENT | An argument is incorrect | | | | | | | | |

E_POSTED The form is posted.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

curses(3X) , **forms(3X)** , **attributes(5)**

NOTES

The header `<form.h>` automatically includes the headers `<eti.h>` and `<curses.h>` .

E_SYSTEM_ERROR System error.
E_BAD_ARGUMENT An argument is incorrect.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`curses(3X)` , `forms(3X)` , `attributes(5)`

NOTES

The header `<form.h>` automatically includes the headers `<eti.h>` and `<curses.h>` .

| | | | | | |
|-----------------------|--|-------------|-------------------------------------|-----------------------|--------------|
| NAME | form_field_buffer, set_field_buffer, field_buffer, set_field_status, field_status, set_max_field – set and get forms field attributes | | | | |
| SYNOPSIS | <pre> cc [flag ...] file ... -lform -lcurses [library ..] #include <form.h> int set_field_buffer(FIELD * field, int buf, char * value); char * field_buffer(FIELD * field, int buf); int set_field_status(FIELD * field, int status); int field_status(FIELD * field); int set_max_field(FIELD * field, int max); </pre> | | | | |
| DESCRIPTION | <p>set_field_buffer() sets buffer <i>buf</i> of <i>field</i> to <i>value</i> . Buffer 0 stores the displayed contents of the field. Buffers other than 0 are application specific and not used by the <code>forms</code> library routines. field_buffer() returns the value of <i>field</i> buffer <i>buf</i> .</p> <p>Every field has an associated status flag that is set whenever the contents of field buffer 0 changes. set_field_status() sets the status flag of <i>field</i> to <i>status</i> . field_status() returns the status of <i>field</i> .</p> <p>set_max_field() sets a maximum growth on a dynamic field, or if <i>max</i>= 0 turns off any maximum growth.</p> | | | | |
| RETURN VALUES | <p>field_buffer() returns <code>NULL</code> on error.</p> <p>field_status() returns <code>TRUE</code> or <code>FALSE</code> .</p> <p>set_field_buffer() , set_field_status() , and set_max_field() return one of the following:</p> <table border="0"> <tr> <td style="padding-right: 20px;">E_OK</td> <td>The function returned successfully.</td> </tr> <tr> <td>E_SYSTEM_ERROR</td> <td>System error</td> </tr> </table> | E_OK | The function returned successfully. | E_SYSTEM_ERROR | System error |
| E_OK | The function returned successfully. | | | | |
| E_SYSTEM_ERROR | System error | | | | |

E_BAD_ARGUMENT An argument is incorrect.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

curses(3X) , **forms(3X)** , **attributes(5)**

NOTES

The header `<form.h>` automatically includes the headers `<eti.h>` and `<curses.h>` .

| NAME | form_field_info, field_info, dynamic_field_info – get forms field characteristics | | | | |
|----------------------|---|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre> cc [flag ...] file ... -lform -lcurses [library ..] #include <form.h> int field_info(FIELD * field, int * rows, int * cols, int * frow, int * fcol, int * nrow, int * nbuf); int dynamic_field_info(FIELD * field, int * drows, int * dcols, int * max); </pre> | | | | |
| DESCRIPTION | <p>field_info() returns the size, position, and other named field characteristics, as defined in the original call to new_field(), to the locations pointed to by the arguments <i>rows</i>, <i>cols</i>, <i>frow</i>, <i>fcol</i>, <i>nrow</i>, and <i>nbuf</i>.</p> <p>dynamic_field_info() returns the actual size of the <i>field</i> in the pointer arguments <i>drows</i>, <i>dcols</i> and returns the maximum growth allowed for <i>field</i> in <i>max</i>. If no maximum growth limit is specified for <i>field</i>, <i>max</i> will contain 0. A field can be made dynamic by turning off the field option <code>O_STATIC</code>.</p> | | | | |
| RETURN VALUES | <p>These routines return one of the following:</p> <p>E_OK The function returned successfully.</p> <p>E_SYSTEM_ERROR System error.</p> <p>E_BAD_ARGUMENT An argument is incorrect.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | curses(3X) , forms(3X) , attributes(5) | | | | |

NOTES | The header `<form.h>` automatically includes the headers `<eti.h>` and `<curses.h>` .

| | |
|----------------------|---|
| NAME | form_field_just, set_field_just, field_just – format the general appearance of forms |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lform -lcurses [<i>library</i> ..] #include <form.h> int set_field_just(FIELD * <i>field</i>, int <i>justification</i>); int field_just(FIELD * <i>field</i>);</pre> |
| DESCRIPTION | <p>set_field_just() sets the justification for <i>field</i> . Justification may be one of:</p> <pre>NO_JUSTIFICATION JUSTIFY_RIGHT JUSTIFY_LEFT JUSTIFY_CENTER</pre> <p>.</p> <p>The field justification will be ignored if <i>field</i> is a dynamic field.</p> <p>field_just() returns the type of justification assigned to <i>field</i> .</p> |
| RETURN VALUES | <p>field_just() returns one of the following:</p> <pre>NO_JUSTIFICATION JUSTIFY_RIGHT JUSTIFY_LEFT JUSTIFY_CENTER .</pre> <p>set_field_just() returns one of the following:</p> |

E_OK The function returned successfully.
E_SYSTEM_ERROR System error.
E_BAD_ARGUMENT An argument is incorrect.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`curses(3X)`, `forms(3X)`, `attributes(5)`

NOTES

The header `<form.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

| | | | | | | | |
|-----------------------|---|-------------|-------------------------------------|--------------------|---|-----------------------|---------------|
| NAME | form_field_new, new_field, dup_field, link_field, free_field – create and destroy forms fields | | | | | | |
| SYNOPSIS | <pre>cc [flag ...] file ... -lform -lcurses [library ..] #include <form.h> FIELD * new_field(int r, int c, int frow, int fcol, int nrow, int ncol); FIELD * dup_field(FIELD * field, int frow, int fcol); FIELD * link_field(FIELD * field, int frow, int fcol); int free_field(FIELD * field);</pre> | | | | | | |
| DESCRIPTION | <p>new_field() creates a new field with <i>r</i> rows and <i>c</i> columns, starting at <i>frow</i> , <i>fcol</i> , in the subwindow of a form. <i>nrow</i> is the number of off-screen rows and <i>nbuf</i> is the number of additional working buffers. This routine returns a pointer to the new field.</p> <p>dup_field() duplicates <i>field</i> at the specified location. All field attributes are duplicated, including the current contents of the field buffers.</p> <p>link_field() also duplicates <i>field</i> at the specified location. However, unlike dup_field() , the new field shares the field buffers with the original field. After creation, the attributes of the new field can be changed without affecting the original field.</p> <p>free_field() frees the storage allocated for <i>field</i> .</p> | | | | | | |
| RETURN VALUES | <p>Routines that return pointers return <code>NULL</code> on error. free_field() returns one of the following:</p> <table border="0"> <tr> <td style="padding-right: 20px;">E_OK</td> <td>The function returned successfully.</td> </tr> <tr> <td>E_CONNECTED</td> <td>The field is already connected to a form.</td> </tr> <tr> <td>E_SYSTEM_ERROR</td> <td>System error.</td> </tr> </table> | E_OK | The function returned successfully. | E_CONNECTED | The field is already connected to a form. | E_SYSTEM_ERROR | System error. |
| E_OK | The function returned successfully. | | | | | | |
| E_CONNECTED | The field is already connected to a form. | | | | | | |
| E_SYSTEM_ERROR | System error. | | | | | | |

E_BAD_ARGUMENT An argument is incorrect.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

curses(3X) , **forms(3X)** , **attributes(5)**

NOTES

The header `<form.h>` automatically includes the headers `<eti.h>` and `<curses.h>` .

| | |
|--------------------|---|
| NAME | form_field_opts, set_field_opts, field_opts_on, field_opts_off, field_opts – forms field option routines |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lform -lcurses [<i>library</i> ..] #include <form.h> int set_field_opts(FIELD * <i>field</i>, OPTIONS <i>opts</i>); int set_field_opts(FIELD * <i>field</i>, OPTIONS <i>opts</i>); int field_opts_on(FIELD * <i>field</i>, OPTIONS <i>opts</i>); int field_opts_off(FIELD * <i>field</i>, OPTIONS <i>opts</i>); OPTIONS field_opts(FIELD * <i>field</i>); </pre> |
| DESCRIPTION | <p>set_field_opts() turns on the named options of <i>field</i> and turns off all remaining options. Options are boolean values that can be OR-ed together.</p> <p>field_opts_on() turns on the named options; no other options are changed.</p> <p>field_opts_off() turns off the named options; no other options are changed.</p> <p>field_opts() returns the options set for <i>field</i>.</p> <p>O_VISIBLE The field is displayed.</p> <p>O_ACTIVE \011The field is visited during processing.</p> <p>O_PUBLIC The field contents are displayed as data is entered.</p> <p>O_EDIT \011The field can be edited.</p> <p>O_WRAP Words not fitting on a line are wrapped to the next line.</p> <p>O_BLANK The whole field is cleared if a character is entered \011in the first position.</p> |

O_AUTOSKIP Skip to the next field when the current field becomes \011full.

O_NULLOK A blank field is considered valid.

O_STATIC The field buffers are fixed in size.

O_PASSOK Validate field only if modified by user.

RETURN VALUES

`set_field_opts`, `field_opts_on` and `field_opts_off` return one of the following:

E_OK The function returned successfully.

E_SYSTEM_ERROR System error.

E_CURRENT The field is the current field.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`curses(3X)`, `forms(3X)`, `attributes(5)`

NOTES

The header `<form.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

| | |
|--------------------|--|
| NAME | form_fieldtype, new_fieldtype, free_fieldtype, set_fieldtype_arg, set_fieldtype_choice, link_fieldtype – forms fieldtype routines |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lform -lcurses [<i>library</i> ..] #include <form.h> FIELDTYPE * new_fieldtype(int (* <i>field_check</i>)(FIELD *, char *), int (* <i>char_check</i>)(int, char *)); int free_fieldtype(FIELDTYPE * <i>fieldtype</i>); int set_fieldtype_arg(FIELDTYPE * <i>fieldtype</i>, char *(* <i>mak_arg</i>)(va_list *), char *(* <i>copy_arg</i>)(char *), void (* <i>free_arg</i>)(char *)); int set_fieldtype_choice(FIELDTYPE * <i>fieldtype</i>, int (* <i>next_choice</i>)(FIELD *, char *), int (* <i>prev_choice</i>)(FIELD *, char *)); FIELDTYPE * link_fieldtype(FIELDTYPE * <i>type1</i>, FIELDTYPE * <i>type2</i>);</pre> |
| DESCRIPTION | <p>new_fieldtype() creates a new field type. The application programmer must write the function <i>field_check</i>, which validates the field value, and the function <i>char_check</i>, which validates each character. free_fieldtype() frees the space allocated for the field type.</p> <p>By associating function pointers with a field type, set_fieldtype_arg() connects to the field type additional arguments necessary for a set_field_type() call. Function <i>mak_arg</i> allocates a structure for the field specific parameters to set_field_type() and returns a pointer to the saved data. Function <i>copy_arg</i> duplicates the structure created by <i>make_arg</i>. Function <i>free_arg</i> frees any storage allocated by <i>make_arg</i> or <i>copy_arg</i>.</p> <p>The form_driver() requests REQ_NEXT_CHOICE and REQ_PREV_CHOICE let the user request the next or previous value of a field type comprising an ordered set of values. set_fieldtype_choice() allows the application programmer to implement these requests for the given field type. It associates with the given field type those application-defined functions that return pointers to the next or previous choice for the field.</p> |

| NAME | form_field_userptr, set_field_userptr, field_userptr – associate application data with forms | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lform -lcurses [<i>library</i> ..] #include <form.h> int set_field_userptr(FIELD * <i>field</i>, char * <i>ptr</i>); char * field_userptr(FIELD * <i>field</i>);</pre> | | | | |
| DESCRIPTION | Every field has an associated user pointer that can be used to store pertinent data. set_field_userptr() sets the user pointer of <i>field</i> . field_userptr() returns the user pointer of <i>field</i> . | | | | |
| RETURN VALUES | <p>field_userptr() returns NULL on error. set_field_userptr() returns one of the following:</p> <p>E_OK The function returned successfully.</p> <p>E_SYSTEM_ERROR System error.</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | curses(3X) , forms(3X) , attributes(5) | | | | |
| NOTES | The header <form.h> automatically includes the headers <eti.h> and <curses.h>. | | | | |

| | | | | | |
|-----------------------|---|-------------|-------------------------------------|-----------------------|---------------|
| NAME | form_field_validation, set_field_type, field_type, field_arg – forms field data type validation | | | | |
| SYNOPSIS | <pre>cc [flag ...] file ... -lform -lcurses [library ..] #include <form.h> int set_field_type(FIELD * field, FIELDTYPE * type, ...); FIELDTYPE * field_type(FIELD * field); char * field_arg(FIELD * field);</pre> | | | | |
| DESCRIPTION | <p>set_field_type() associates the specified field type with <i>field</i>. Certain field types take additional arguments. <code>TYPE_ALNUM</code>, for instance, requires one, the minimum width specification for the field. The other predefined field types are: <code>TYPE_ALPHA</code>, <code>TYPE_ENUM</code>, <code>TYPE_INTEGER</code>, <code>TYPE_NUMERIC</code>, and <code>TYPE_REGEX</code>.</p> <p>field_type() returns a pointer to the field type of <i>field</i>. <code>NULL</code> is returned if no field type is assigned.</p> <p>field_arg() returns a pointer to the field arguments associated with the field type of <i>field</i>. <code>NULL</code> is returned if no field type is assigned.</p> | | | | |
| RETURN VALUES | <p>field_type() and field_arg() return <code>NULL</code> on error.</p> <p>set_field_type() returns one of the following:</p> <table border="0"> <tr> <td style="padding-right: 20px;">E_OK</td> <td>The function returned successfully.</td> </tr> <tr> <td>E_SYSTEM_ERROR</td> <td>System error.</td> </tr> </table> | E_OK | The function returned successfully. | E_SYSTEM_ERROR | System error. |
| E_OK | The function returned successfully. | | | | |
| E_SYSTEM_ERROR | System error. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `curses(3X)`, `forms(3X)`, `attributes(5)`

NOTES The header `<form.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

| | |
|--------------------|--|
| NAME | form_hook, set_form_init, form_init, set_form_term, form_term, set_field_init, field_init, set_field_term, field_term – assign application-specific routines for invocation by forms |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lform -lcurses [<i>library</i> ..] #include <form.h> int set_form_init(FORM * form, void (*func)(FORM*)); void (*form_init)(FORM * form); int set_form_term(FORM * form, void (*func)(FORM*)); void (*form_term)(FORM * form); int set_field_init(FORM * form, void (*func)(FORM*)); void (*field_init)(FORM * form); int set_field_term(FORM * form, void (*func)(FORM*)); void (*field_term)(FORM * form);</pre> |
| DESCRIPTION | <p>These routines allow the programmer to assign application specific routines to be executed automatically at initialization and termination points in the forms application. The user need not specify any application-defined initialization or termination routines at all, but they may be helpful for displaying messages or page numbers and other chores.</p> <p>set_form_init() assigns an application-defined initialization function to be called when the <i>form</i> is posted and just after a page change. form_init() returns a pointer to the initialization function, if any.</p> <p>set_form_term() assigns an application-defined function to be called when the <i>form</i> is unposted and just before a page change. form_term() returns a pointer to the function, if any.</p> |

set_field_init() assigns an application-defined function to be called when the *form* is posted and just after the current field changes. **field_init()** returns a pointer to the function, if any.

set_field_term() assigns an application-defined function to be called when the *form* is unposted and just before the current field changes. **field_term()** returns a pointer to the function, if any.

RETURN VALUES

Routines that return pointers always return `NULL` on error. Routines that return an integer return one of the following:

`E_OK` - The function returned successfully.
`E_SYSTEM_ERROR` - System error.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

courses(3X), **forms(3X)**, **attributes(5)**

NOTES

The header `<form.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

| NAME | form_new, new_form, free_form – create and destroy forms | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lform -lcurses [<i>library</i> ..] #include <form.h> FORM * new_form(FIELD ** <i>fields</i>); int free_form(FORM * <i>form</i>);</pre> | | | | |
| DESCRIPTION | <p>new_form() creates a new form connected to the designated fields and returns a pointer to the form.</p> <p>free_form() disconnects the <i>form</i> from its associated field pointer array and deallocates the space for the form.</p> | | | | |
| RETURN VALUES | <p>new_form() always returns NULL on error. free_form() returns one of the following:</p> <pre>E_OK - The function returned successfully. E_BAD_ARGUMENT - An argument is incorrect. E_POSTED - The form is posted.</pre> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | curses(3X) , forms(3X) , attributes(5) | | | | |
| NOTES | The header <form.h> automatically includes the headers <eti.h> and <curses.h> . | | | | |

| NAME | form_new_page, set_new_page, new_page - forms pagination | | | | |
|----------------------|---|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lform -lcurses [<i>library</i> ..] #include <form.h> int set_new_page(FIELD * <i>field</i>, int <i>bool</i>); int new_page(FIELD * <i>field</i>);</pre> | | | | |
| DESCRIPTION | <p>set_new_page() marks <i>field</i> as the beginning of a new page on the form.</p> <p>new_page() returns a boolean value indicating whether or not <i>field</i> begins a new page of the form.</p> | | | | |
| RETURN VALUES | <p>new_page returns TRUE or FALSE .</p> <p>set_new_page() returns one of the following:</p> <p>E_OK - The function returned successfully. E_CONNECTED - The field is already connected to a form. E_SYSTEM_ERROR - System error.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" data-bbox="500 1461 1396 1549"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | curses(3X) , forms(3X) , attributes(5) | | | | |
| NOTES | The header <form.h> automatically includes the headers <eti.h> and <curses.h> . | | | | |

| | |
|----------------------|---|
| NAME | form_opts, set_form_opts, form_opts_on, form_opts_off – forms option routines |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lform -lcurses [<i>library</i> ..] #include <form.h> int set_form_opts(FORM * <i>form</i>, OPTIONS <i>opts</i>); int form_opts_on(FORM * <i>form</i>, OPTIONS <i>opts</i>); int form_opts_off(FORM * <i>form</i>, OPTIONS <i>opts</i>); OPTIONS form_opts(FORM * <i>form</i>); </pre> |
| DESCRIPTION | <p>set_form_opts() turns on the named options for <i>form</i> and turns off all remaining options. Options are boolean values which can be OR-ed together.</p> <p>form_opts_on() turns on the named options; no other options are changed.</p> <p>form_opts_off() turns off the named options; no other options are changed.</p> <p>form_opts() returns the options set for <i>form</i> .</p> <p>Form Options:</p> <pre> O_NL_OVERLOAD\011Overload the REQ_NEW_LINE form driver request. O_BS_OVERLOAD\011Overload the REQ_DEL_PREV form driver request. </pre> |
| RETURN VALUES | <p>set_form_opts() , form_opts_on() , and form_opts_off() return one of the following:</p> <pre> E_OK - The function returned successfully. E_SYSTEM_ERROR - System error. </pre> |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`curses(3X)`, `forms(3X)`, `attributes(5)`

NOTES

The header `<form.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

| | |
|----------------------|--|
| NAME | form_page, set_form_page, set_current_field, current_field, field_index – set forms current page and field |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lform -lcurses [<i>library</i> ..] #include <form.h> int set_form_page(FORM * <i>form</i>, int <i>page</i>); int form_page(FORM * <i>form</i>); int set_current_field(FORM * <i>form</i>, FIELD * <i>field</i>); FIELD * current_field(FORM* <i>form</i>); int field_index(FIELD * <i>field</i>);</pre> |
| DESCRIPTION | <p>set_form_page() sets the page number of <i>form</i> to <i>page</i> . form_page() returns the current page number of <i>form</i> .</p> <p>set_current_field() sets the current field of <i>form</i> to <i>field</i> . current_field() returns a pointer to the current field of <i>form</i> .</p> <p>field_index() returns the index in the field pointer array of <i>field</i> .</p> |
| RETURN VALUES | <p>form_page() returns -1 on error.</p> <p>current_field() returns NULL on error.</p> <p>field_index() returns -1 on error.</p> <p>set_form_page() and set_current_field() return one of the following:</p> <pre>E_OK - The function returned successfully. E_SYSTEM_ERROR - System error. E_BAD_ARGUMENT - An argument is incorrect. E_BAD_STATE - The routine was called from an \011\011initialization or termination function. E_INVALID_FIELD - The field contents are invalid. E_REQUEST_DENIED - The form driver request failed.</pre> |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`curses(3X)`, `forms(3X)`, `attributes(5)`

NOTES

The header `<form.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

| NAME | form_post, post_form, unpost_form – write or erase forms from associated subwindows | | | | |
|----------------------|---|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lform -lcurses [<i>library</i> ..] #include <form.h> int post_form(FORM * form); int unpost_form(FORM * form);</pre> | | | | |
| DESCRIPTION | <p>post_form() writes <i>form</i> into its associated subwindow. The application programmer must use <code>curses</code> library routines to display the form on the physical screen or call update_panels() if the <code>panels</code> library is being used.</p> <p>unpost_form() erases <i>form</i> from its associated subwindow.</p> | | | | |
| RETURN VALUES | <p>These routines return one of the following:</p> <pre>E_OK - The function returned successfully. E_SYSTEM_ERROR - System error. E_BAD_ARGUMENT - An argument is incorrect. E_POSTED - The form is posted. E_NOT_POSTED - The form is not posted. E_NO_ROOM - The form does not fit in the subwindow. E_BAD_STATE - The routine was called from an \011\011initialization or termination function. E_NOT_CONNECTED - The field is not connected to a form.</pre> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |

SEE ALSO | `curses(3X)`, `forms(3X)`, `panel_update(3X)`, `panels(3X)`,
`attributes(5)`

NOTES | The header `<form.h>` automatically includes the headers `<eti.h>` and
`<curses.h>`.

| NAME | forms – character based forms package | | | | | | | | | | | | | | | | | | | | | | |
|--|---|---------------------------|------------------|----------------------|---------------|-------------------|---------------|--------------------|---------------|------------------|--------------------|---------------------------|---------------------|------------------|---------------------------|-------------------|---------------------------|---------------------|-----------------------|--------------------|----------------|-------------------|---------------------------|
| SYNOPSIS | <code>#include <form.h></code> | | | | | | | | | | | | | | | | | | | | | | |
| DESCRIPTION | <p>The <code>form</code> library is built using the <code>curses</code> library, and any program using <code>forms</code> routines must call one of the <code>curses</code> initialization routines such as <code>initscr</code>. A program using these routines must be compiled with <code>-lform</code> and <code>-lcurses</code> on the <code>cc</code> command line.</p> <p>The <code>forms</code> package gives the applications programmer a terminal-independent method of creating and customizing forms for user-interaction. The <code>forms</code> package includes: field routines, which are used to create and customize fields, link fields and assign field types; <code>fieldtype</code> routines, which are used to create new field types for validating fields; and form routines, which are used to create and customize forms, assign pre/post processing functions, and display and interact with forms.</p> | | | | | | | | | | | | | | | | | | | | | | |
| Current Default Values for Field Attributes | <p>The <code>forms</code> package establishes initial current default values for field attributes. During field initialization, each field attribute is assigned the current default value for that attribute. An application can change or retrieve a current default attribute value by calling the appropriate set or retrieve routine with a <code>NULL</code> field pointer. If an application changes a current default field attribute value, subsequent fields created using <code>new_field()</code> will have the new default attribute value. (The attributes of previously created fields are not changed if a current default attribute value is changed.)</p> | | | | | | | | | | | | | | | | | | | | | | |
| Routine Name Index | <p>The following table lists each <code>forms</code> routine and the name of the manual page on which it is described.</p> <table border="0"> <thead> <tr> <th style="text-align: left;">forms Routine Name</th> <th style="text-align: left;">Manual Page Name</th> </tr> </thead> <tbody> <tr> <td>current_field</td> <td>form_page(3X)</td> </tr> <tr> <td>data_ahead</td> <td>form_data(3X)</td> </tr> <tr> <td>data_behind</td> <td>form_data(3X)</td> </tr> <tr> <td>dup_field</td> <td>form_field_new(3X)</td> </tr> <tr> <td>dynamic_field_info</td> <td>form_field_info(3X)</td> </tr> <tr> <td>field_arg</td> <td>form_field_validation(3X)</td> </tr> <tr> <td>field_back</td> <td>form_field_attributes(3X)</td> </tr> <tr> <td>field_buffer</td> <td>form_field_buffer(3X)</td> </tr> <tr> <td>field_count</td> <td>form_field(3X)</td> </tr> <tr> <td>field_fore</td> <td>form_field_attributes(3X)</td> </tr> </tbody> </table> | forms Routine Name | Manual Page Name | current_field | form_page(3X) | data_ahead | form_data(3X) | data_behind | form_data(3X) | dup_field | form_field_new(3X) | dynamic_field_info | form_field_info(3X) | field_arg | form_field_validation(3X) | field_back | form_field_attributes(3X) | field_buffer | form_field_buffer(3X) | field_count | form_field(3X) | field_fore | form_field_attributes(3X) |
| forms Routine Name | Manual Page Name | | | | | | | | | | | | | | | | | | | | | | |
| current_field | form_page(3X) | | | | | | | | | | | | | | | | | | | | | | |
| data_ahead | form_data(3X) | | | | | | | | | | | | | | | | | | | | | | |
| data_behind | form_data(3X) | | | | | | | | | | | | | | | | | | | | | | |
| dup_field | form_field_new(3X) | | | | | | | | | | | | | | | | | | | | | | |
| dynamic_field_info | form_field_info(3X) | | | | | | | | | | | | | | | | | | | | | | |
| field_arg | form_field_validation(3X) | | | | | | | | | | | | | | | | | | | | | | |
| field_back | form_field_attributes(3X) | | | | | | | | | | | | | | | | | | | | | | |
| field_buffer | form_field_buffer(3X) | | | | | | | | | | | | | | | | | | | | | | |
| field_count | form_field(3X) | | | | | | | | | | | | | | | | | | | | | | |
| field_fore | form_field_attributes(3X) | | | | | | | | | | | | | | | | | | | | | | |

| | |
|-----------------------|---------------------------|
| field_index | form_page(3X) |
| field_info | form_field_info(3X) |
| field_init | form_hook(3X) |
| field_just | form_field_just(3X) |
| field_opts | form_field_opts(3X) |
| field_opts_off | form_field_opts(3X) |
| field_opts_on | form_field_opts(3X) |
| field_pad | form_field_attributes(3X) |
| field_status | form_field_buffer(3X) |
| field_term | form_hook(3X) |
| field_type | form_field_validation(3X) |
| field_userptr | form_field_userptr(3X) |
| form_driver | form_driver(3X) |
| form_fields | form_field(3X) |
| form_init | form_hook(3X) |
| form_opts | form_opts(3X) |
| form_opts_off | form_opts(3X) |
| form_opts_on | form_opts(3X) |
| form_page | form_page(3X) |
| form_sub | form_win(3X) |
| form_term | form_hook(3X) |
| form_userptr | form_userptr(3X) |
| form_win | form_win(3X) |
| free_field | form_field_new(3X) |
| free_fieldtype | form_fieldtype(3X) |
| free_form | form_new(3X) |
| link_field | form_field_new(3X) |

| | |
|-----------------------------|---------------------------|
| link_fieldtype | form_fieldtype(3X) |
| move_field | form_field(3X) |
| new_field | form_field_new(3X) |
| new_fieldtype | form_fieldtype(3X) |
| new_form | form_new(3X) |
| new_page | form_new_page(3X) |
| pos_form_cursor | form_cursor(3X) |
| post_form | form_post(3X) |
| scale_form | form_win(3X) |
| set_current_field | form_page(3X) |
| set_field_back | form_field_attributes(3X) |
| set_field_buffer | form_field_buffer(3X) |
| set_field_fore | form_field_attributes(3X) |
| set_field_init | form_hook(3X) |
| set_field_just | form_field_just(3X) |
| set_field_opts | form_field_opts(3X) |
| set_field_pad | form_field_attributes(3X) |
| set_field_status | form_field_buffer(3X) |
| set_field_term | form_hook(3X) |
| set_field_type | form_field_validation(3X) |
| set_field_userptr | form_field_userptr(3X) |
| set_fieldtype_arg | form_fieldtype(3X) |
| set_fieldtype_choice | form_fieldtype(3X) |
| set_form_fields | form_field(3X) |
| set_form_init | form_hook(3X) |
| set_form_opts | form_opts(3X) |
| set_form_page | form_page(3X) |

| | |
|-------------------------|-----------------------|
| set_form_sub | form_win(3X) |
| set_form_term | form_hook(3X) |
| set_form_userptr | form_userptr(3X) |
| set_form_win | form_win(3X) |
| set_max_field | form_field_buffer(3X) |
| set_new_page | form_new_page(3X) |
| unpost_form | form_post(3X) |

RETURN VALUES

Routines that return a pointer always return `NULL` on error. Routines that return an integer return one of the following:

| | |
|--------------------------------|--|
| <code>E_OK</code> | The function returned successfully. |
| <code>E_CONNECTED</code> | The field is already connected to a form. |
| <code>E_SYSTEM_ERROR</code> | System error. |
| <code>E_BAD_ARGUMENT</code> | An argument is incorrect. |
| <code>E_CURRENT</code> | The field is the current field. |
| <code>E_POSTED</code> | The form is posted. |
| <code>E_NOT_POSTED</code> | The form is not posted. |
| <code>E_INVALID_FIELD</code> | The field contents are invalid. |
| <code>E_NOT_CONNECTED</code> | The field is not connected to a form. |
| <code>E_NO_ROOM</code> | The form does not fit in the subwindow. |
| <code>E_BAD_STATE</code> | The routine was called from an initialization or termination function. |
| <code>E_REQUEST_DENIED</code> | The form driver request failed. |
| <code>E_UNKNOWN_COMMAND</code> | An unknown request was passed to the form driver. |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `curses(3X)`, `attributes(5)` and 3X pages whose names begin "form_" for detailed routine descriptions.

NOTES The header `<form.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

| NAME | form_userptr, set_form_userptr – associate application data with forms | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lform -lcurses [<i>library</i> ..] #include <form.h> int set_form_userptr(FORM * <i>form</i>, char * <i>ptr</i>); char * form_userptr(FORM * <i>form</i>);</pre> | | | | |
| DESCRIPTION | Every form has an associated user pointer that can be used to store pertinent data. set_form_userptr() sets the user pointer of <i>form</i> . form_userptr() returns the user pointer of <i>form</i> . | | | | |
| RETURN VALUES | <p>form_userptr() returns NULL on error. set_form_userptr() returns one of the following:</p> <p>E_OK – The function returned successfully. E_SYSTEM_ERROR – System error.</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | curses(3X) , forms(3X) , attributes(5) | | | | |
| NOTES | The header <form.h> automatically includes the headers <eti.h> and <curses.h>. | | | | |

| | |
|----------------------|--|
| NAME | form_win, set_form_win, set_form_sub, form_sub, scale_form – forms window and subwindow association routines |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lform -lcurses [<i>library</i> ..] #include <form.h> int set_form_win(FORM * <i>form</i>, WINDOW * <i>win</i>); WINDOW * form_win(FORM * <i>form</i>); int set_form_sub(FORM * <i>form</i>, WINDOW * <i>sub</i>); WINDOW * form_sub(FORM * <i>form</i>); int scale_form(FORM * <i>form</i>, int * <i>rows</i>, int * <i>cols</i>);</pre> |
| DESCRIPTION | <p>set_form_win() sets the window of <i>form</i> to <i>win</i> . form_win() returns a pointer to the window associated with <i>form</i> .</p> <p>set_form_sub() sets the subwindow of <i>form</i> to <i>sub</i> . form_sub() returns a pointer to the subwindow associated with <i>form</i> .</p> <p>scale_form() returns the smallest window size necessary for the subwindow of <i>form</i> . <i>rows</i> and <i>cols</i> are pointers to the locations used to return the number of rows and columns for the form.</p> |
| RETURN VALUES | <p>Routines that return pointers always return NULL on error. Routines that return an integer return one of the following:</p> <pre>E_OK - The function returned successfully. E_SYSTEM_ERROR - System error. E_BAD_ARGUMENT - An argument is incorrect. E_NOT_CONNECTED - The field is not connected to a form. E_POSTED - The form is posted.</pre> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `curses(3X)`, `forms(3X)`, `attributes(5)`

NOTES The header `<form.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

| | |
|--------------------|--|
| NAME | fpgetround, fpsetround, fpgetmask, fpsetmask, fpgetsticky, fpsetsticky – IEEE floating-point environment control |
| SYNOPSIS | <pre>#include <ieeefp.h> fp_rnd fpgetround(void); fp_rnd fpsetround(fp_rnd rnd_dir); fp_except fpgetmask(void); fp_except fpsetmask(fp_except mask); fp_except fpgetsticky(void); fp_except fpsetsticky(fp_except sticky);</pre> |
| DESCRIPTION | <p>There are five floating-point exceptions:</p> <ul style="list-style-type: none"> ■ divide-by-zero, ■ overflow, ■ underflow, ■ imprecise (inexact) result, and ■ invalid operation. <p>When a floating-point exception occurs, the corresponding sticky bit is set (1), and if the mask bit is enabled (1), the trap takes place. These routines let the user change the behavior on occurrence of any of these exceptions, as well as change the rounding mode for floating-point operations.</p> <p>The <i>mask</i> argument is formed by the logical OR operation of the following floating-point exception masks:</p> <pre>FP_X_INV /* invalid operation exception */ FP_X_OFLO /* overflow exception */ FP_X_UFLO /* underflow exception */ FP_X_DZ /* divide-by-zero exception */ FP_X_IMP /* imprecise (loss of precision) */</pre> <p>The following floating-point rounding modes are passed to <code>fpsetround</code> and returned by <code>fpgetround()</code>.</p> <pre>FP_RN /* round to nearest representative number */ FP_RP /* round to plus infinity */</pre> |

```
FP_RM      /* round to minus infinity */
FP_RZ      /* round to zero (truncate) */
```

The default environment is rounding mode set to nearest (`FP_RN`) and all traps disabled.

The `fpsetsticky()` function modifies all sticky flags. The `fpsetmask()` function changes all mask bits. The `fpgetmask()` function clears the sticky bit corresponding to any exception being enabled.

RETURN VALUES

The `fpgetround()` function returns the current rounding mode.

The `fpsetround()` function sets the rounding mode and returns the previous rounding mode.

The `fpgetmask()` function returns the current exception masks.

The `fpsetmask()` function sets the exception masks and returns the previous setting.

The `fpgetsticky()` function returns the current exception sticky flags.

The `fpsetsticky()` function sets (clears) the exception sticky flags and returns the previous setting.

USAGE

The C programming language requires truncation (round to zero) for floating point to integral conversions. The current rounding mode has no effect on these conversions.

The sticky bit must be cleared to recover from the trap and proceed. If the sticky bit is not cleared before the next trap occurs, a wrong exception type may be signaled.

Individual bits may be examined using the constants defined in `<ieeefp.h>`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`isnan(3C)` , `attributes(5)`

| | |
|---|---|
| NAME | fputc, putc, putc_unlocked, putchar, putchar_unlocked, putw – put a byte on a stream |
| SYNOPSIS | <pre>#include <stdio.h> int fputc(int c, FILE * stream); int putc(int c, FILE * stream); int putc_unlocked(int c, FILE * stream); int putchar(int c); int putchar_unlocked(int c); int putw(int w, FILE * stream);</pre> |
| DESCRIPTION | |
| fputc() | <p>The fputc() function writes the byte specified by <i>c</i> (converted to an <code>unsigned char</code>) to the output stream pointed to by <i>stream</i>, at the position indicated by the associated file-position indicator for the stream (if defined), and advances the indicator appropriately. If the file cannot support positioning requests, or if the stream was opened with append mode, the byte is appended to the output stream.</p> <p>The <code>st_ctime</code> and <code>st_mtime</code> fields of the file will be marked for update between the successful execution of fputc() and the next successful completion of a call to fflush(3S) or fclose(3S) on the same stream or a call to exit(3C) or abort(3C).</p> |
| putc() | <p>The putc() routine behaves like fputc(), except that it is implemented as a macro. It runs faster than fputc(), but it takes up more space per invocation and its name cannot be passed as an argument to a function call.</p> |
| putchar() | <p>The call <code>putchar(c)</code> is equivalent to <code>putc(c , stdout)</code>. The putchar() routine is implemented as a macro.</p> |
| putc_unlocked() and putchar_unlocked() | <p>The putc_unlocked() and putchar_unlocked() routines are variants of putc() and putchar(), respectively, that do not lock the stream. It is the caller's responsibility to acquire the stream lock before calling these routines and releasing the lock afterwards; see flockfile(3S) and stdio(3S). These routines are implemented as macros.</p> |
| putw() | <p>The putw() function writes the word (that is, type <code>int</code>) <i>w</i> to the output <i>stream</i> (at the position at which the file offset, if defined, is pointing). The size of a</p> |

word is the size of a type `int` and varies from machine to machine. The `putw()` function neither assumes nor causes special alignment in the file.

The `st_ctime` and `st_mtime` fields of the file will be marked for update between the successful execution of `putw()` and the next successful completion of a call to `fflush(3S)` or `fclose(3S)` on the same stream or a call to `exit(3C)` or `abort(3C)`.

RETURN VALUES

Upon successful completion, `fputc()`, `putc()`, `putc_unlocked()`, `putchar()`, and `putchar_unlocked()` return the value that was written. Otherwise, these functions return EOF, the error indicator for the stream is set, and `errno` is set to indicate the error.

Upon successful completion, `putw()` returns 0. Otherwise, it returns a non-zero value, sets the error indicator for the associated *stream*, and sets `errno` to indicate the error.

An unsuccessful completion will occur, for example, if the file associated with *stream* is not open for writing or if the output file cannot grow.

ERRORS

The `fputc()`, `putc()`, `putc_unlocked()`, `putchar()`, `putchar_unlocked()`, and `putw()` functions will fail if either the *stream* is unbuffered or the *stream*'s buffer needs to be flushed, and:

- | | |
|---------------|---|
| EAGAIN | The <code>O_NONBLOCK</code> flag is set for the file descriptor underlying <i>stream</i> and the process would be delayed in the write operation. |
| EBADF | The file descriptor underlying <i>stream</i> is not a valid file descriptor open for writing. |
| EFBIG | An attempt was made to write to a file that exceeds the maximum file size or the process' file size limit. |
| EFBIG | The file is a regular file and an attempt was made to write at or beyond the offset maximum. |
| EINTR | The write operation was terminated due to the receipt of a signal, and no data was transferred. |
| EIO | A physical I/O error has occurred, or the process is a member of a background process group attempting to write to its controlling terminal, <code>TOSTOP</code> is set, the process is neither ignoring nor blocking <code>SIGTTOU</code> and the process group of the process is orphaned. This error may also be returned under implementation-dependent conditions. |

ENOSPC There was no free space remaining on the device containing the file.

EPIPE An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal will also be sent to the thread.

The **fputc()** , **putc()** , **putc_unlocked()** , **putchar()** , **putchar_unlocked()** , and **putw()** functions may fail if:

ENOMEM Insufficient storage space is available.

ENXIO A request was made of a non-existent device, or the request was outside the capabilities of the device.

USAGE

Functions exist for the **putc()** , **putc_unlocked()** , **putchar()** , and **putchar_unlocked()** macros. To get the function form, the macro name must be undefined (for example, `#undef putc`).

When the macro forms are used, **putc()** and **putc_unlocked()** evaluate the *stream* argument more than once. In particular, `putc(c , *f++);` does not work sensibly. The **fputc()** function should be used instead when evaluating the *stream* argument has side effects.

Because of possible differences in word length and byte ordering, files written using **putw()** are implementation-dependent, and possibly cannot be read using **getw(3S)** by a different application or by the same application running in a different environment.

The **putw()** function is inherently byte stream oriented and is not tenable in the context of either multibyte character streams or wide-character streams. Application programmers are encouraged to use one of the character-based output functions instead.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|------------------|
| MT-Level | See NOTES below. |

SEE ALSO

getrlimit(2) , **ulimit(2)** **write(2)** , **intro(3)** , **abort(3C)** , **exit(3C)** , **fclose(3S)** , **ferror(3S)** , **fflush(3S)** , **flockfile(3S)** , **fopen(3B)** , **printf(3S)** , **putc(3S)** , **puts(3S)** , **setbuf(3S)** , **stdio(3S)** , **attributes(5)**

NOTES

The **fputc()** , **putc()** , **putchar()** , and **putw()** routines are MT-Safe in multithreaded applications. The **putc_unlocked()** and **putchar_unlocked()** routines are unsafe in multithreaded applications.

| | |
|----------------------|--|
| NAME | fputwc, putwc, putwchar – put wide-character code on a stream |
| SYNOPSIS | <pre>#include <stdio.h> #include <wchar.h> wint_t fputwc(wchar_t wc, FILE* stream); wint_t putwc(wchar_t wc, FILE* stream); #include <wchar.h> wint_t putwchar(wchar_t wc);</pre> |
| DESCRIPTION | |
| fputwc() | <p>The fputwc() function writes the character corresponding to the wide-character code <i>wc</i> to the output stream pointed to by <i>stream</i>, at the position indicated by the associated file-position indicator for the stream (if defined), and advances the indicator appropriately. If the file cannot support positioning requests, or if the stream was opened with append mode, the character is appended to the output stream. If an error occurs while writing the character, the shift state of the output file is left in an undefined state.</p> <p>The <i>st_ctime</i> and <i>st_mtime</i> fields of the file will be marked for update between the successful execution of fputwc() and the next successful completion of a call to fflush(3S) or fclose(3S) on the same stream or a call to exit(2) or abort(3C).</p> |
| putwc() | The putwc() function is equivalent to fputwc() , except that it is implemented as a macro. |
| putwchar() | The call putwchar(wc) is equivalent to putwc(wc, stdout) . The putwchar() routine is implemented as a macro. |
| RETURN VALUES | Upon successful completion, fputwc() , putwc() , and putwchar() return <i>wc</i> . Otherwise, they return WEOF , the error indicator for the stream is set, and errno is set to indicate the error. |
| ERRORS | <p>The fputwc(), putwc(), and putwchar() functions will fail if either the stream is unbuffered or data in the <i>stream</i>'s buffer needs to be written, and:</p> <p>EAGAIN The O_NONBLOCK flag is set for the file descriptor underlying <i>stream</i> and the process would be delayed in the write operation.</p> <p>EBADF The file descriptor underlying <i>stream</i> is not a valid file descriptor open for writing.</p> |

- EFBIG** An attempt was made to write to a file that exceeds the maximum file size or the process's file size limit; or the file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.
 - EINTR** The write operation was terminated due to the receipt of a signal, and no data was transferred.
 - EIO** A physical I/O error has occurred, or the process is a member of a background process group attempting to write to its controlling terminal, `TOSTOP` is set, the process is neither ignoring nor blocking `SIGTTOU`, and the process group of the process is orphaned.
 - ENOSPC** There was no free space remaining on the device containing the file.
 - EPIPE** An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A `SIGPIPE` signal will also be sent to the process.
- The `fputc()`, `putc()`, and `putwchar()` functions may fail if:
- ENOMEM** Insufficient storage space is available.
 - ENXIO** A request was made of a non-existent device, or the request was outside the capabilities of the device.
 - EILSEQ** The wide-character code `wc` does not correspond to a valid character.

USAGE

Functions exist for the `putc()` and `putwchar()` macros. To get the function form, the macro name must be undefined (for example, `#undef putc`).

When the macro form is used, `putc()` evaluates the *stream* argument more than once. In particular, `putc(wc, *f++)` does not work sensibly. The `fputc()` function should be used instead when evaluating the *stream* argument has side effects.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | `exit(2)`, `ulimit(2)`, `abort(3C)`, `fclose(3S)`, `ferror(3S)`, `fflush(3S)`,
`fopen(3S)`, `setbuf(3S)`, `attributes(5)`

| NAME | fputws – put wide character string on a stream | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <stdio.h> #include <wchar.h> int fputws(const wchar_t *s, FILE *stream);</pre> | | | | |
| DESCRIPTION | <p>The fputws() function writes a character string corresponding to the (null-terminated) wide character string pointed to by <i>ws</i> to the stream pointed to by <i>stream</i>. No character corresponding to the terminating null wide-character code is written, nor is a NEWLINE character appended.</p> <p>The <i>st_ctime</i> and <i>st_mtime</i> fields of the file will be marked for update between the successful execution of fputws() and the next successful completion of a call to fflush(3S) or fclose(3S) on the same stream or a call to exit(2) or abort(3C).</p> | | | | |
| RETURN VALUES | <p>Upon successful completion, fputws() returns a non-negative value. Otherwise, it returns <code>-1</code>, sets an error indicator for the stream, and sets <i>errno</i> to indicate the error.</p> | | | | |
| ERRORS | Refer to fputwc(3S) . | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | exit(2) , abort(3C) , fclose(3S) , fflush(3S) , fopen(3S) , fputwc(3S) , attributes(5) | | | | |

| | |
|----------------------|---|
| NAME | fread, fwrite – buffered binary input/output |
| SYNOPSIS | <pre>#include <stdio.h> size_t fread(void * ptr, size_t size, size_t nitems, FILE * stream); size_t fwrite(const void * ptr, size_t size, size_t nitems, FILE * stream);</pre> |
| DESCRIPTION | <p>The fread() function reads into an array pointed to by <i>ptr</i> up to <i>nitems</i> items of data from <i>stream</i>, where an item of data is a sequence of bytes (not necessarily terminated by a null byte) of length <i>size</i>. It stops reading bytes if an end-of-file or error condition is encountered while reading <i>stream</i>, or if <i>nitems</i> items have been read. It increments the data pointer in <i>stream</i> to point to the byte following the last byte read if there is one. It does not change the contents of <i>stream</i>. It returns the number of items read.</p> <p>The fwrite() function writes to the named output <i>stream</i> at most <i>nitems</i> items of data from the array pointed to by <i>ptr</i>, where an item of data is a sequence of bytes (not necessarily terminated by a null byte) of length <i>size</i>. It stops writing when it has written <i>nitems</i> items of data or if an error condition is encountered on <i>stream</i>. It does not change the contents of the array pointed to by <i>ptr</i>. It increments the data pointer in <i>stream</i> by the number of bytes written and returns the number of items written.</p> <p>A call to fwrite() in buffered mode may return success even though the underlying call to write(2) fails. This can cause unpredictable results. Use either the write() function or the fwrite() function in unbuffered mode. See setvbuf(3S).</p> <p>The ferror() or feof() routines must be used to distinguish between an error condition and end-of-file condition. See ferror(3S).</p> |
| RETURN VALUES | <p>The fread() function returns the number of items read. The fwrite() function returns the number of items written.</p> <p>If <i>size</i> or <i>nitems</i> is 0, then fread() and fwrite() return 0 and do not effect the state of <i>stream</i>.</p> <p>If an error occurs, fread() and fwrite() return 0 and set the error indicator for <i>stream</i>.</p> |
| ERRORS | <p>The fread() function will fail if data needs to be read and:</p> <p>E_OVERFLOW The file is a regular file, <i>size</i> is greater than 0, the starting position is before the end-of-file, and an attempt was made to read at or beyond the offset maximum associated with the corresponding <i>stream</i>.</p> <p>The fwrite() function will fail if either the <i>stream</i> is unbuffered or the <i>stream</i>'s buffer needed to be flushed and:</p> |

EFBIG The file is a regular file and an attempt was made to write at or beyond the offset maximum.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

read(2), **write(2)**, **fclose(3S)**, **ferror(3S)**, **fopen(3S)**, **getc(3S)**, **gets(3S)**, **printf(3S)**, **putc(3S)**, **puts(3S)**, **scanf(3S)**, **setvbuf(3S)**, **stdio(3S)**, **attributes(5)**

| | |
|----------------------|--|
| NAME | freopen – open a stream |
| SYNOPSIS | <pre>#include <stdio.h> FILE *freopen(const char *filename, const char *mode, FILE *stream);</pre> |
| DESCRIPTION | <p>The freopen() function first attempts to flush the stream and close any file descriptor associated with <i>stream</i>. Failure to flush or close the file successfully is ignored. The error and end-of-file indicators for the stream are cleared.</p> <p>The freopen() function opens the file whose pathname is the string pointed to by <i>filename</i> and associates the stream pointed to by <i>stream</i> with it. The <i>mode</i> argument is used just as in fopen(3S).</p> <p>The original stream is closed regardless of whether the subsequent open succeeds.</p> <p>After a successful call to the freopen() function, the orientation of the stream is cleared and the associated <code>mbstate_t</code> object is set to describe an initial conversion state.</p> <p>The largest value that can be represented correctly in an object of type <code>off_t</code> will be established as the offset maximum in the open file description.</p> |
| RETURN VALUES | Upon successful completion, freopen() returns the value of <i>stream</i> . Otherwise, a null pointer is returned and <code>errno</code> is set to indicate the error. |
| ERRORS | <p>The freopen() function will fail if:</p> <p>EACCES Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by <i>mode</i> are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created.</p> <p>EINTR A signal was caught during freopen().</p> <p>EISDIR The named file is a directory and <i>mode</i> requires write access.</p> <p>ELOOP Too many symbolic links were encountered in resolving <i>path</i>.</p> <p>EMFILE There are <code>OPEN_MAX</code> file descriptors currently open in the calling process.</p> <p>ENAMETOOLONG The length of the <i>filename</i> exceeds <code>PATH_MAX</code> or a pathname component is longer than <code>NAME_MAX</code>.</p> |

| | |
|--|---|
| ENFILE | The maximum allowable number of files is currently open in the system. |
| ENOENT | A component of <i>filename</i> does not name an existing file or <i>filename</i> is an empty string. |
| ENOSPC | The directory or file system that would contain the new file cannot be expanded, the file does not exist, and it was to be created. |
| ENOTDIR | A component of the path prefix is not a directory. |
| ENXIO | The named file is a character special or block special file, and the device associated with this special file does not exist. |
| EOVERFLOW | The current value of the file position cannot be represented correctly in an object of type <code>off_t</code> . |
| EROFS | The named file resides on a read-only file system and <i>mode</i> requires write access. |
| The freopen() function may fail if: | |
| EINVAL | The value of the <i>mode</i> argument is not valid. |
| ENAMETOOLONG | Pathname resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code> . |
| ENOMEM | Insufficient storage space is available. |
| ENXIO | A request was made of a non-existent device, or the request was outside the capabilities of the device. |
| ETXTBSY | The file is a pure procedure (shared text) file that is being executed and <i>mode</i> requires write access. |

USAGE

The **freopen()** function is typically used to attach the preopened *streams* associated with `stdin`, `stdout` and `stderr` to other files. By default `stderr` is unbuffered, but the use of **freopen()** will cause it to become buffered or line-buffered.

The **freopen()** function has a transitional interface for 64-bit file offsets. See `1f64(5)`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO `fclose(3S)`, `fdopen(3S)`, `fopen(3S)`, `stdio(3S)`, `attributes(5)`, `lf64(5)`

NAME frexp – extract mantissa and exponent from double precision number

SYNOPSIS #include <math.h>

```
double frexp(double num, int *exp);
```

DESCRIPTION The **frexp()** function breaks a floating-point number into a normalized fraction and an integral power of 2. It stores the integer exponent in the `int` object pointed to by `exp`.

RETURN VALUES The **frexp()** function returns the value `x`, such that `x` is a `double` with magnitude in the interval $[\frac{1}{2}, 1)$ or 0, and `num` equals `x` times 2 raised to the power `*exp`.

If `num` is 0, both parts of the result are 0.

If `num` is NaN, NaN is returned and the value of `*exp` is unspecified.

If `num` is $\pm\text{Inf}$, `num` is returned and the value of `*exp` is unspecified.

USAGE An application wishing to check for error situations should set `errno` to 0 before calling **frexp()**. If `errno` is non-zero on return, or the return value is NaN, an error has occurred.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **isnan(3M)**, **ldexp(3C)**, **modf(3C)**, **attributes(5)**

| | |
|----------------------|--|
| NAME | fseek, fseeko – reposition a file-position indicator in a stream |
| SYNOPSIS | <pre>#include <stdio.h> int fseek(FILE * <i>stream</i>, long <i>offset</i>, int <i>whence</i>); int fseeko(FILE * <i>stream</i>, off_t <i>offset</i>, int <i>whence</i>);</pre> |
| DESCRIPTION | <p>The fseek() function sets the file-position indicator for the stream pointed to by <i>stream</i> . The fseeko() function is identical to fseek() except for the type of <i>offset</i> .</p> <p>The new position, measured in bytes from the beginning of the file, is obtained by adding <i>offset</i> to the position specified by <i>whence</i> , whose values are defined in <code><stdio.h></code> as follows:</p> <p>SEEK_SET Set position equal to <i>offset</i> bytes.</p> <p>SEEK_CUR Set position to current location plus <i>offset</i> .</p> <p>SEEK_END Set position to EOF plus <i>offset</i> .</p> <p>If the stream is to be used with wide character input/output functions, <i>offset</i> must either be 0 or a value returned by an earlier call to ftell(3S) on the same stream and <i>whence</i> must be SEEK_SET .</p> <p>A successful call to fseek() clears the end-of-file indicator for the stream and undoes any effects of ungetc(3S) and ungetwc(3S) on the same stream. After an fseek() call, the next operation on an update stream may be either input or output.</p> <p>If the most recent operation, other than ftell(3S) , on a given stream is fflush(3S) , the file offset in the underlying open file description will be adjusted to reflect the location specified by fseek() .</p> <p>The fseek() function allows the file-position indicator to be set beyond the end of existing data in the file. If data is later written at this point, subsequent reads of data in the gap will return bytes with the value 0 until data is actually written into the gap.</p> <p>The value of the file offset returned by fseek() on devices which are incapable of seeking is undefined.</p> <p>If the stream is writable and buffered data had not been written to the underlying file, fseek() will cause the unwritten data to be written to the file and mark the <code>st_ctime</code> and <code>st_mtime</code> fields of the file for update.</p> |
| RETURN VALUES | The fseek() and fseeko() functions return 0 on success; otherwise, they returned -1 and set <code>errno</code> to indicate the error. |

| | |
|------------------|--|
| ERRORS | <p>The fseek() and fseeko() functions will fail if, either the <i>stream</i> is unbuffered or the <i>stream</i>'s buffer needed to be flushed, and the call to fseek() or fseeko() causes an underlying lseek(2) or write(2) to be invoked:</p> |
| EAGAIN | The <code>O_NONBLOCK</code> flag is set for the file descriptor and the process would be delayed in the write operation. |
| EBADF | The file descriptor underlying the stream file is not open for writing or the stream's buffer needed to be flushed and the file is not open. |
| EFBIG | An attempt was made to write a file that exceeds the maximum file size or the process's file size limit, or the file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream. |
| EINTR | The write operation was terminated due to the receipt of a signal, and no data was transferred. |
| EINVAL | The <i>whence</i> argument is invalid. The resulting file-position indicator would be set to a negative value. |
| EIO | A physical I/O error has occurred; or the process is a member of a background process group attempting to perform a write(2) operation to its controlling terminal, <code>TOSTOP</code> is set, the process is neither ignoring nor blocking <code>SIGTTOU</code> , and the process group of the process is orphaned. |
| ENOSPC | There was no free space remaining on the device containing the file. |
| EPIPE | The file descriptor underlying <i>stream</i> is associated with a pipe or FIFO. |
| EPIPE | An attempt was made to write to a pipe or FIFO that is not open for reading by any process. A <code>SIGPIPE</code> signal will also be sent to the process. |
| ENXIO | A request was made of a non-existent device, or the request was outside the capabilities of the device. |
| | The fseek() function will fail if: |
| Eoverflow | The resulting file offset would be a value which cannot be represented correctly in an object of type <code>long</code> . |
| | The fseeko() function will fail if: |
| Eoverflow | The resulting file offset would be a value which cannot be represented correctly in an object of type <code>off_t</code> . |

USAGE Although on the UNIX system an offset returned by **ftell()** or **ftello()** (see **ftell(3S)**) is measured in bytes, and it is permissible to seek to positions relative to that offset, portability to non-UNIX systems requires that an offset be used by **fseek()** directly. Arithmetic may not meaningfully be performed on such an offset, which is not necessarily measured in bytes.

The **fseeko()** function has a transitional interface for 64-bit file offsets. See **lf64(5)**.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **getrlimit(2)**, **ulimit(2)**, **fopen(3B)**, **ftell(3S)**, **rewind(3S)**, **ungetc(3S)**, **ungetwc(3S)**, **attributes(5)**, **lf64(5)**

| NAME | fsetpos – reposition a file pointer in a stream | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | #include <stdio.h> int fsetpos (FILE * <i>stream</i> , const fpos_t * <i>pos</i>); | | | | |
| DESCRIPTION | The fsetpos() function sets the file position indicator for the stream pointed to by <i>stream</i> according to the value of the object pointed to by <i>pos</i> , which must be a value obtained from an earlier call to fgetpos(3S) on the same stream. A successful call to fsetpos() function clears the end-of-file indicator for the stream and undoes any effects of ungetc(3S) on the same stream. After an fsetpos() call, the next operation on an update stream may be either input or output. | | | | |
| RETURN VALUES | The fsetpos() function returns 0 if it succeeds; otherwise it returns a non-zero value and sets <code>errno</code> to indicate the error. | | | | |
| ERRORS | The fsetpos() function may fail if: EBADF The file descriptor underlying <i>stream</i> is not valid. ESPIPE The file descriptor underlying <i>stream</i> is associated with a pipe, a FIFO, or a socket. | | | | |
| USAGE | The fsetpos() function has a transitional interface for 64-bit file offsets. See lf64(5) . | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | lseek(2) , fgetpos(3S) , fopen(3S) , fseek(3S) , ftell(3S) , rewind(3S) , ungetc(3S) , attributes(5) , lf64(5) | | | | |

| | |
|----------------------|--|
| NAME | fsync – synchronize a file’s in-memory state with that on the physical medium |
| SYNOPSIS | <pre>#include <unistd.h> int fsync(int <i>fil</i>des);</pre> |
| DESCRIPTION | <p>The fsync() function moves all modified data and attributes of the file descriptor <i>fil</i>des to a storage device. When fsync() returns, all in-memory modified copies of buffers associated with <i>fil</i>des have been written to the physical medium. The fsync() function is different from sync(), which schedules disk I/O for all files but returns before the I/O completes. The fsync() function forces all outstanding data operations to synchronized file integrity completion (see fcntl(5) definition of <code>O_SYNC</code>.)</p> |
| RETURN VALUES | <p>Upon successful completion, 0 is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error.</p> |
| ERRORS | <p>The fsync() function will fail if:</p> <p>EBADF The <i>fil</i>des argument is not a valid file descriptor.</p> <p>EINTR A signal was caught during execution of the fsync() function.</p> <p>EIO An I/O error occurred while reading from or writing to the file system.</p> <p>ENOSPC There was no free space remaining on the device containing the file.</p> <p>ETIMEDOUT Remote connection timed out. This occurs when the file is on an NFS file system mounted with the <i>soft</i> option. See mount_nfs(1M).</p> |
| USAGE | <p>The fsync() function should be used by applications that require that a file be in a known state. For example, an application that contains a simple transaction facility might use fsync() to ensure that all changes to a file or files caused by a given transaction were recorded on a storage medium.</p> <p>The manner in which the data reach the physical medium depends on both implementation and hardware. The fsync() function returns when notified by the device driver that the write has taken place.</p> |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------|
| MT-Level | Async-Signal-Safe |

SEE ALSO

`mount_nfs(1M)`, `sync(2)`, `fdatasync(3R)`, `attributes(5)`, `fcntl(5)`

NAME | ftell, ftello – return a file offset in a stream

SYNOPSIS | #include <stdio.h>

| long **ftell**(FILE * *stream*);

| off_t **ftello**(FILE * *stream*);

DESCRIPTION | The **ftell()** function obtains the current value of the file-position indicator for the stream pointed to by *stream* . The **ftello()** function is identical to **ftell()** except for the return type.

RETURN VALUES | Upon successful completion, the **ftell()** and **ftello()** functions return the current value of the file-position indicator for the stream measured in bytes from the beginning of the file. Otherwise, they return `-1` and sets `errno` to indicate the error.

ERRORS | The **ftell()** and **ftello()** functions will fail if:

EBADF | The file descriptor underlying *stream* is not an open file descriptor.

ESPIPE | The file descriptor underlying *stream* is associated with a pipe, a FIFO, or a socket.

| The **ftell()** function will fail if:

EOVERFLOW | The current file offset cannot be represented correctly in an object of type `long` .

| The **ftello()** function will fail if:

EOVERFLOW | The current file offset cannot be represented correctly in an object of type `off_t` .

USAGE | The **ftello()** function has a transitional interface for 64-bit file offsets. See **1f64(5)** .

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **lseek(2)** , **fopen(3S)** , **fseek(3S)** , **attributes(5)** , **1f64(5)**

| | |
|----------------------|---|
| NAME | ftime – get date and time |
| SYNOPSIS | <pre>#include <sys/timeb.h> int ftime(struct timeb *tp);</pre> |
| DESCRIPTION | <p>The ftime() function sets the <code>time</code> and <code>millitm</code> members of the <code>timeb</code> structure pointed to by <code>tp</code>. The structure is defined in <code><sys/timeb.h></code> and contains the following members:</p> <pre>time_t time; unsigned short millitm; short timezone; short dstflag;</pre> <p>The <code>time</code> and <code>millitm</code> members contain the seconds and milliseconds portions, respectively, of the current time in seconds since 00:00:00 UTC (Coordinated Universal Time), January 1, 1970.</p> <p>The <code>timezone</code> member contains the local time zone. The <code>dstflag</code> member contains a flag that, if non-zero, indicates that Daylight Saving time applies locally during the appropriate part of the year.</p> <p>The contents of the <code>timezone</code> and <code>dstflag</code> members of <code>tp</code> after a call to ftime() are unspecified.</p> |
| RETURN VALUES | Upon successful completion, the ftime() function returns 0. Otherwise <code>-1</code> is returned. |
| ERRORS | No errors are defined. |
| USAGE | <p>For portability to implementations conforming to earlier versions of this document, <code>time(2)</code> is preferred over this function.</p> <p>The millisecond value usually has a granularity greater than one due to the resolution of the system clock. Depending on any granularity (particularly a granularity of one) renders code non-portable.</p> |
| SEE ALSO | <code>date(1)</code> , <code>time(2)</code> , <code>ctime(3C)</code> , <code>gettimeofday(3C)</code> , <code>timezone(4)</code> |

| | |
|----------------------|---|
| NAME | ftok – generate an IPC key |
| SYNOPSIS | <pre>#include <sys/ipc.h> key_t ftok(const char *path, int id);</pre> |
| DESCRIPTION | <p>The ftok() function returns a key based on <i>path</i> and <i>id</i> that is usable in subsequent calls to msgget(2), semget(2) and shmget(2). The <i>path</i> argument must be the pathname of an existing file that the process is able to stat(2).</p> <p>The ftok() function will return the same key value for all paths that name the same file, when called with the same <i>id</i> value, and will return different key values when called with different <i>id</i> values or with paths that name different files existing on the same file system at the same time.</p> <p>If the file named by <i>path</i> is removed while still referred to by a key, a call to ftok() with the same <i>path</i> and <i>id</i> returns an error. If the same file is recreated, then a call to ftok() with the same <i>path</i> and <i>id</i> is likely to return a different key.</p> <p>Only the low order 8-bits of <i>id</i> are significant. The behavior of ftok() is unspecified if these bits are 0.</p> |
| RETURN VALUES | Upon successful completion, ftok() returns a key. Otherwise, ftok() returns $(key_t)-1$ and sets <code>errno</code> to indicate the error. |
| ERRORS | <p>The ftok() function will fail if:</p> <p>EACCES Search permission is denied for a component of the path prefix.</p> <p>ELOOP Too many symbolic links were encountered in resolving <i>path</i>.</p> <p>ENAMETOOLONG The length of the <i>path</i> argument exceeds <code>{PATH_MAX}</code> or a pathname component is longer than <code>{NAME_MAX}</code>.</p> <p>ENOENT A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.</p> <p>ENOTDIR A component of the path prefix is not a directory.</p> <p>The ftok() function may fail if:</p> <p>ENAMETOOLONG Pathname resolution of a symbolic link produced an intermediate result whose length exceeds <code>{PATH_MAX}</code>.</p> |
| USAGE | For maximum portability, <i>id</i> should be a single-byte character. |

Another way to compose keys is to include the project ID in the most significant byte and to use the remaining portion as a sequence number. There are many other ways to form keys, but it is necessary for each system to define standards for forming them. If some standard is not adhered to, it will be possible for unrelated processes to unintentionally interfere with each other's operation. It is still possible to interfere intentionally. Therefore, it is strongly suggested that the most significant byte of a key in some sense refer to a project so that keys do not conflict across a given system.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`msgget(2)`, `semget(2)`, `shmget(2)`, `stat(2)`, `attributes(5)`

| | |
|--------------------|---|
| NAME | ftw, nftw – walk a file tree |
| SYNOPSIS | <pre>#include <ftw.h> int ftw(const char * <i>path</i>, int (* <i>fn</i>)(const char *, const struct stat *, int), int <i>depth</i>); int nftw(const char * <i>path</i>, int (* <i>fn</i>)(const char *, const struct stat *, int, struct FTW*), int <i>depth</i>, int <i>flags</i>);</pre> |
| DESCRIPTION | <p>The ftw() function recursively descends the directory hierarchy rooted in <i>path</i>. For each object in the hierarchy, ftw() calls the user-defined function <i>fn</i>, passing it a pointer to a null-terminated character string containing the name of the object, a pointer to a <i>stat</i> structure (see stat(2)) containing information about the object, and an integer. Possible values of the integer, defined in the <i><ftw.h></i> header, are:</p> <p>FTW_F The object is a file.</p> <p>FTW_D The object is a directory.</p> <p>FTW_DNR The object is a directory that cannot be read. Descendants of the directory will not be processed.</p> <p>FTW_NS The stat() function failed on the object because of lack of appropriate permission or the object is a symbolic link that points to a non-existent file. The <i>stat</i> buffer passed to <i>fn</i> is undefined.</p> <p>The ftw() function visits a directory before visiting any of its descendants.</p> <p>The tree traversal continues until the tree is exhausted, an invocation of <i>fn</i> returns a non-zero value, or some error is detected within ftw() (such as an I/O error). If the tree is exhausted, ftw() returns 0. If <i>fn</i> returns a non-zero value, ftw() stops its tree traversal and returns whatever value was returned by <i>fn</i>.</p> <p>The nftw() function is similar to ftw() except that it takes the additional argument <i>flags</i>, whose possible values are:</p> <p>FTW_PHYS Physical walk, does not follow symbolic links. Otherwise, nftw() will follow links but will not walk down any path that crosses itself.</p> <p>FTW_MOUNT The walk will not cross a mount point.</p> <p>FTW_DEPTH All subdirectories will be visited before the directory itself.</p> <p>FTW_CHDIR The walk will change to each directory before reading it. The nftw() function calls <i>fn</i> with four arguments at each file and directory. The first argument is the pathname of the object, the second is a pointer to the</p> |

`stat` buffer, the third is an integer giving additional information, and the fourth is a `struct FTW` that contains the following members:

```
int    base;
int    level;
```

The `base` member is the offset into the pathname of the base name of the object. The `level` member indicates the depth relative to the rest of the walk, where the root level is zero.

The values of the third argument are as follows:

| | |
|----------------------|--|
| <code>FTW_F</code> | The object is a file. |
| <code>FTW_D</code> | The object is a directory. |
| <code>FTW_DP</code> | The object is a directory and subdirectories have been visited. |
| <code>FTW_SL</code> | The object is a symbolic link. |
| <code>FTW_SLN</code> | The object is a symbolic link that points to a non-existent file. |
| <code>FTW_DNR</code> | The object is a directory that cannot be read. The user-defined function <code>fn</code> will not be called for any of its descendants. |
| <code>FTW_NS</code> | The <code>stat()</code> function failed on the object because of lack of appropriate permission. The <code>stat</code> buffer passed to <code>fn</code> is undefined. The <code>stat</code> function fail for a reason other than lack of appropriate permission. <code>EACCES</code> is considered an error and <code>nftw()</code> will return <code>-1</code> . |

Both `ftw()` and `nftw()` use one file descriptor for each level in the tree. The `depth` argument limits the number of file descriptors so used. If `depth` is zero or negative, the effect is the same as if it were 1. It must not be greater than the number of file descriptors currently available for use. The `ftw()` function will run faster if `depth` is at least as large as the number of levels in the tree. When `ftw()` and `nftw()` return, they close any file descriptors they have opened; they do not close any file descriptors that may have been opened by `fn`.

RETURN VALUES

If successful, `ftw()` and `nftw()` return 0. If either function detects an error other than `EACCES`, it returns `-1` and sets `errno` to indicate the error.

USAGE

Because `ftw()` is recursive, it is possible for it to terminate with a memory fault when applied to very deep file structures.

The **ftw()** function uses **malloc(3C)** to allocate dynamic storage during its operation. If **ftw()** is forcibly terminated, such as by **longjmp(3C)** being executed by *fn* or an interrupt routine, **ftw()** will not have a chance to free that storage, so it will remain permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have *fn* return a non-zero value at its next invocation.

The **ftw()** and **nftw()** functions have transitional interfaces for 64-bit file offsets. See **lf64(5)**.

The **ftw()** function is safe in multithreaded applications. The **nftw()** function is safe in multithreaded applications when the `FTW_CHDIR` flag is not set.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------------|
| MT-Level | Safe with exceptions. |

SEE ALSO

stat(2), **longjmp(3C)**, **malloc(3C)**, **attributes(5)**, **lf64(5)**

| NAME | fwide – set stream orientation | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <stdio.h> #include <wchar.h> int fwide(FILE *stream, int mode);</pre> | | | | |
| DESCRIPTION | <p>The fwide() function determines the orientation of the stream pointed to by <i>stream</i>. If <i>mode</i> is greater than 0, the function first attempts to make the stream wide-orientated. If <i>mode</i> is less than 0, the function first attempts to make the stream byte-orientated. Otherwise, <i>mode</i> is 0 and the function does not alter the orientation of the stream.</p> <p>If the orientation of the stream has already been determined, fwide() does not change it.</p> <p>Because no return value is reserved to indicate an error, an application wishing to check for error situations should set <code>errno</code> to 0, then call fwide(), then check <code>errno</code> and if it is non-zero, assume an error has occurred.</p> | | | | |
| RETURN VALUES | The fwide() function returns a value greater than 0 if, after the call, the stream has wide-orientation, a value less than 0 if the stream has byte-orientation, or 0 if the stream has no orientation. | | | | |
| ERRORS | <p>The fwide() function may fail if:</p> <p>EBADF The <i>stream</i> argument is not a valid stream.</p> | | | | |
| USAGE | A call to fwide() with <i>mode</i> set to 0 can be used to determine the current orientation of a stream. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | attributes(5) | | | | |

| | |
|--------------------|---|
| NAME | fwprintf, wprintf, swprintf – print formatted wide-character output |
| SYNOPSIS | <pre>#include <stdio.h> #include <wchar.h> int fwprintf(FILE * stream, const wchar_t * format, ...); int wprintf(const wchar_t * format, < ...>); int swprintf(wchar_t * s, size_t n, const wchar_t * format, ...);</pre> |
| DESCRIPTION | <p>The fwprintf() function places output on the named output <i>stream</i> . The wprintf() function places output on the standard output stream <code>stdout</code> . The swprintf() function places output followed by the null wide-character in consecutive wide-characters starting at <i>*s</i> ; no more than <i>n</i> wide-characters are written, including a terminating null wide-character, which is always added (unless <i>n</i> is zero).</p> <p>Each of these functions converts, formats and prints its arguments under control of the <i>format</i> wide-character string. The <i>format</i> is composed of zero or more directives: <i>ordinary wide-characters</i> , which are simply copied to the output stream and <i>conversion specifications</i> , each of which results in the fetching of zero or more arguments. The results are undefined if there are insufficient arguments for the <i>format</i> . If the <i>format</i> is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.</p> <p>Conversions can be applied to the <i>n</i> th argument after the <i>format</i> in the argument list, rather than to the next unused argument. In this case, the conversion wide-character % (see below) is replaced by the sequence % <i>n</i> \$, where <i>n</i> is a decimal integer in the range [1, NL_ARGMAX], giving the position of the argument in the argument list. This feature provides for the definition of format wide-character strings that select arguments in an order appropriate to specific languages (see the EXAMPLES section).</p> <p>In format wide-character strings containing the % <i>n</i> \$ form of conversion specifications, numbered arguments in the argument list can be referenced from the format wide-character string as many times as required.</p> <p>In format wide-character strings containing the % form of conversion specifications, each argument in the argument list is used exactly once.</p> <p>All forms of the fwprintf() functions allow for the insertion of a language-dependent radix character in the output string, output as a wide-character value. The radix character is defined in the program's locale (category LC_NUMERIC). In the POSIX locale, or in a locale where the radix character is not defined, the radix character defaults to a period (.) .</p> <p>Each conversion specification is introduced by the % wide-character or by the wide-character sequence % <i>n</i> \$, after which the following appear in sequence:</p> |

- Zero or more *flags* (in any order), which modify the meaning of the conversion specification.
- An optional minimum *field width*. If the converted value has fewer wide-characters than the field width, it will be padded with spaces by default on the left; it will be padded on the right, if the left-adjustment flag (–), described below, is given to the field width. The field width takes the form of an asterisk (*), described below, or a decimal integer.
- An optional *precision* that gives the minimum number of digits to appear for the `d`, `i`, `o`, `u`, `x`, and `X` conversions; the number of digits to appear after the radix character for the `e`, `E`, and `f` conversions; the maximum number of significant digits for the `g` and `G` conversions; or the maximum number of wide-characters to be printed from a string in `s` conversions. The precision takes the form of a period (.) followed by either an asterisk (*), described below, or an optional decimal digit string, where a null digit string is treated as 0. If a precision appears with any other conversion wide-character, the behavior is undefined.
- An optional `l` (ell) specifying that a following `c` conversion wide-character applies to a `wint_t` argument; an optional `l` specifying that a following `s` conversion wide-character applies to a `wchar_t` argument; an optional `h` specifying that a following `d`, `i`, `o`, `u`, `x`, and `X` conversion wide-character applies to a type `short int` or type `unsigned short int` argument (the argument will have been promoted according to the integral promotions, and its value will be converted to type `short int` or `unsigned short int` before printing); an optional `h` specifying that a following `n` conversion wide-character applies to a pointer to a type `short int` argument; an optional `l` (ell) specifying that a following `d`, `i`, `o`, `u`, `x`, and `X` conversion wide-character applies to a type `long int` or `unsigned long int` argument; an optional `l` (ell) specifying that a following `n` conversion wide-character applies to a pointer to a type `long int` argument; or an optional `L` specifying that a following `e`, `E`, `f`, `g`, or `G` conversion wide-character applies to a type `long double` argument. If an `h`, `l`, or `L` appears with any other conversion wide-character, the behavior is undefined.
- A *conversion wide-character* that indicates the type of conversion to be applied.

A field width, or precision, or both, may be indicated by an asterisk (*). In this case an argument of type `int` supplies the field width or precision. Arguments specifying field width, or precision, or both must appear in that order before the argument, if any, to be converted. A negative field width is taken as a – flag followed by a positive field width. A negative precision is taken as if the precision were omitted. In format wide-character strings containing the `% n $` form of a conversion specification, a field width or precision may be indicated

by the sequence `* m $`, where `m` is a decimal integer in the range `[1, NL_ARGMAX]` giving the position in the argument list (after the format argument) of an integer argument containing the field width or precision, for example:

```
wprintf(L"%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```

The *format* can contain either numbered argument specifications (that is, `% n $` and `* m $`), or unnumbered argument specifications (that is, `%` and `*`), but normally not both. The only exception to this is that `%%` can be mixed with the `% n $` form. The results of mixing numbered and unnumbered argument specifications in a *format* wide-character string are undefined. When numbered argument specifications are used, specifying the N th argument requires that all the leading arguments, from the first to the $(N-1)$ th, are specified in the format wide-character string.

The flag wide-characters and their meanings are:

| | |
|--------------|--|
| ' | The integer portion of the result of a decimal conversion (<code>%i</code> , <code>%d</code> , <code>%u</code> , <code>%f</code> , <code>%g</code> , or <code>%G</code>) will be formatted with thousands' grouping wide-characters. For other conversions the behavior is undefined. The non-monetary grouping wide-character is used. |
| – | The result of the conversion will be left-justified within the field. The conversion will be right-justified if this flag is not specified. |
| + | The result of a signed conversion will always begin with a sign (<code>+</code> or <code>–</code>). The conversion will begin with a sign only when a negative value is converted if this flag is not specified. |
| space | If the first wide-character of a signed conversion is not a sign or if a signed conversion results in no wide-characters, a space will be prefixed to the result. This means that if the space and <code>+</code> flags both appear, the space flag will be ignored. |
| # | This flag specifies that the value is to be converted to an alternative form. For <code>o</code> conversion, it increases the precision (if necessary) to force the first digit of the result to be 0. For <code>x</code> or <code>X</code> conversions, a non-zero result will have <code>0x</code> (or <code>0X</code>) prefixed to it. For <code>e</code> , <code>E</code> , <code>f</code> , <code>g</code> , or <code>G</code> conversions, the result will always contain a radix character, even if no digits follow it. Without this flag, a radix character appears in the result of |

these conversions only if a digit follows it. For `g` and `G` conversions, trailing zeros will *not* be removed from the result as they normally are. For other conversions, the behavior is undefined.

- `0` For `d`, `i`, `o`, `u`, `x`, `X`, `e`, `E`, `f`, `g`, and `G` conversions, leading zeros (following any indication of sign or base) are used to pad to the field width; no space padding is performed. If the `0` and `-` flags both appear, the `0` flag will be ignored. For `d`, `i`, `o`, `u`, `x`, and `X` conversions, if a precision is specified, the `0` flag will be ignored. If the `0` and `'` flags both appear, the grouping wide-characters are inserted before zero padding. For other conversions, the behavior is undefined.

The conversion wide-characters and their meanings are:

- `d`, `i` The `int` argument is converted to a signed decimal in the style `[-] dddd`. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting 0 with an explicit precision of 0 is no wide-characters.
- `o` The unsigned `int` argument is converted to unsigned octal format in the style `dddd`. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting 0 with an explicit precision of 0 is no wide-characters.
- `u` The unsigned `int` argument is converted to unsigned decimal format in the style `dddd`. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting 0 with an explicit precision of 0 is no wide-characters.
- `x` The unsigned `int` argument is converted to unsigned hexadecimal format in the style `dddd`; the letters `abcdef` are used. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting 0 with an explicit precision of 0 is no wide-characters.
- `X` Behaves the same as the `x` conversion wide-character except that letters `ABCDEF` are used instead of `abcdef`.

f The `double` argument is converted to decimal notation in the style [-] *ddd.ddd*, where the number of digits after the radix character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is explicitly 0 and no # flag is present, no radix character appears. If a radix character appears, at least one digit appears before it. The value is rounded to the appropriate number of digits.

The **fwprintf()** family of functions may make available wide-character string representations for infinity and NaN.

e , E The `double` argument is converted in the style [-] *d.ddd e ± dd*, where there is one digit before the radix character (which is non-zero if the argument is non-zero) and the number of digits after it is equal to the precision; if the precision is missing, it is taken as 6; if the precision is 0 and no # flag is present, no radix character appears. The value is rounded to the appropriate number of digits. The **E** conversion wide-character will produce a number with **E** instead of **e** introducing the exponent. The exponent always contains at least two digits. If the value is 0, the exponent is 0.

The **fwprintf()** family of functions may make available wide-character string representations for infinity and NaN.

g , G The `double` argument is converted in the style **f** or **e** (or in the style **E** in the case of a **G** conversion wide-character), with the precision specifying the number of significant digits. If an explicit precision is 0, it is taken as 1. The style used depends on the value converted; style **e** (or **E**) will be used only if the exponent resulting from such a conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional portion of the result; a radix character appears only if it is followed by a digit.

The **fwprintf()** family of functions may make available wide-character string representations for infinity and NaN.

c If no **l** (**ell**) qualifier is present, the `int` argument is converted to a wide-character as if by calling the **btowc**(3C) function and the resulting wide-character is written. Otherwise the `wint_t` argument is converted to `wchar_t`, and written.

s If no **l** (**ell**) qualifier is present, the argument must be a pointer to a character array containing a character sequence beginning in the initial shift state. Characters from the array are converted as if by repeated calls to the **mbrtowc**(3C) function, with the conversion state described

by an `mbstate_t` object initialized to zero before the first character is converted, and written up to (but not including) the terminating null wide-character. If the precision is specified, no more than that many wide-characters are written. If the precision is not specified or is greater than the size of the array, the array must contain a null wide-character.

If an `l` (ell) qualifier is present, the argument must be a pointer to an array of type `wchar_t`. Wide characters from the array are written up to (but not including) a terminating null wide-character. If no precision is specified or is greater than the size of the array, the array must contain a null wide-character. If a precision is specified, no more than that many wide-characters are written.

P The argument must be a pointer to `void`. The value of the pointer is converted to a sequence of printable wide-characters.

n The argument must be a pointer to an integer into which is written the number of wide-characters written to the output so far by this call to one of the **fwprintf()** functions. No argument is converted.

C Same as `lc`.

S Same as `ls`.

% Output a % wide-character; no argument is converted. The entire conversion specification must be `%%`.

If a conversion specification does not match one of the above forms, the behavior is undefined.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by **fwprintf()** and **wprintf()** are printed as if **fputwc(3S)** had been called.

The `st_ctime` and `st_mtime` fields of the file will be marked for update between the call to a successful execution of **fwprintf()** or **wprintf()** and the next successful completion of a call to **fflush(3S)** or **fclose(3S)** on the same stream or a call to **exit(3C)** or **abort(3C)**.

RETURN VALUES

Upon successful completion, these functions return the number of wide-characters transmitted excluding the terminating null wide-character in the case of **swprintf()** or a negative value if an output error was encountered.

ERRORS

For the conditions under which **fwprintf()** and **wprintf()** will fail and may fail, refer to **fputwc(3S)**.

In addition, all forms of **fwprintf()** may fail if:

EILSEQ A wide-character code that does not correspond to a valid character has been detected.

EINVAL There are insufficient arguments.

In addition, **wprintf()** and **fwprintf()** may fail if:

ENOMEM Insufficient storage space is available.

EXAMPLES

EXAMPLE 1 Print language-dependent date and time format.

To print the language-independent date and time format, the following statement could be used:

```
wprintf(format, weekday, month, day, hour, min);
```

For American usage, *format* could be a pointer to the wide-character string:

```
L"%s, %s %d, %d:%.2d\
"
```

producing the message:

```
Sunday, July 3, 10:02
```

whereas for German usage, *format* could be a pointer to the wide-character string:

```
L"%1$s, %3$d. %2$s, %4$d:%5$.2d\
"
```

producing the message:

```
Sonntag, 3. Juli, 10:02
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |

SEE ALSO

btowc(3C), **fputwc(3S)**, **fwscanf(3S)**, **mbrtowc(3C)**, **setlocale(3C)**, **attributes(5)**

NOTES

The **fwprintf()**, **wprintf()**, and **swprintf()** functions can be used safely in multithreaded applications, as long as **setlocale(3C)** is not being called to change the locale.

| | |
|--------------------|---|
| NAME | fwscanf, wscanf, swscanf – convert formatted wide-character input |
| SYNOPSIS | <pre>#include <stdio.h> #include <wchar.h> int fwscanf(FILE * stream, const wchar_t * format, ...); int wscanf(const wchar_t * format, ...); int swscanf(const wchar_t * s, const wchar_t * format, ...);</pre> |
| DESCRIPTION | <p>The fwscanf() function reads from the named input <i>stream</i>. The wscanf() function reads from the standard input stream <code>stdin</code>. The swscanf() function reads from the wide-character string <i>s</i>. Each function reads wide-characters, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control wide-character string <i>format</i> described below, and a set of <i>pointer</i> arguments indicating where the converted input should be stored. The result is undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.</p> <p>Conversions can be applied to the <i>n</i> th argument after the <i>format</i> in the argument list, rather than to the next unused argument. In this case, the conversion wide-character <code>%</code> (see below) is replaced by the sequence <code>% n \$</code>, where <i>n</i> is a decimal integer in the range <code>[1, NL_ARGMAX]</code>. This feature provides for the definition of format wide-character strings that select arguments in an order appropriate to specific languages. In format wide-character strings containing the <code>% n \$</code> form of conversion specifications, it is unspecified whether numbered arguments in the argument list can be referenced from the format wide-character string more than once.</p> <p>The <i>format</i> can contain either form of a conversion specification, that is, <code>%</code> or <code>% n \$</code>, but the two forms cannot normally be mixed within a single <i>format</i> wide-character string. The only exception to this is that <code>%%</code> or <code>%%*</code> can be mixed with the <code>% n \$</code> form.</p> <p>The fwscanf() function in all its forms allows for detection of a language-dependent radix character in the input string, encoded as a wide-character value. The radix character is defined in the program's locale (category <code>LC_NUMERIC</code>). In the POSIX locale, or in a locale where the radix character is not defined, the radix character defaults to a period (<code>.</code>).</p> <p>The format is a wide-character string composed of zero or more directives. Each directive is composed of one of the following: one or more white-space wide-characters (space, tab, newline, vertical-tab or form-feed characters); an ordinary wide-character (neither <code>%</code> nor a white-space character); or a conversion specification. Each conversion specification is introduced by a <code>%</code> or the sequence <code>% n \$</code> after which the following appear in sequence:</p> |

- An optional assignment-suppressing character `*`.
- An optional non-zero decimal integer that specifies the maximum field width.
- An optional size modifier `h`, `l` (ell), or `L` indicating the size of the receiving object. The conversion wide-characters `c`, `s`, and `[]` must be preceded by `l` (ell) if the corresponding argument is a pointer to `wchar_t` rather than a pointer to a character type. The conversion wide-characters `d`, `i`, and `n` must be preceded by `h` if the corresponding argument is a pointer to `short int` rather than a pointer to `int`, or by `l` (ell) if it is a pointer to `long int`. Similarly, the conversion wide-characters `o`, `u`, and `x` must be preceded by `h` if the corresponding argument is a pointer to `unsigned short int` rather than a pointer to `unsigned int`, or by `l` (ell) if it is a pointer to `unsigned long int`. The conversion wide-characters `e`, `f`, and `g` must be preceded by `l` (ell) if the corresponding argument is a pointer to `double` rather than a pointer to `float`, or by `L` if it is a pointer to `long double`. If an `h`, `l` (ell), or `L` appears with any other conversion wide-character, the behavior is undefined.
- A conversion wide-character that specifies the type of conversion to be applied. The valid conversion wide-characters are described below.

The **fwscanf()** functions execute each directive of the format in turn. If a directive fails, as detailed below, the function returns. Failures are described as input failures (due to the unavailability of input bytes) or matching failures (due to inappropriate input).

A directive composed of one or more white-space wide-characters is executed by reading input until no more valid input can be read, or up to the first wide-character which is not a white-space wide-character, which remains unread.

A directive that is an ordinary wide-character is executed as follows. The next wide-character is read from the input and compared with the wide-character that comprises the directive; if the comparison shows that they are not equivalent, the directive fails, and the differing and subsequent wide-characters remain unread.

A directive that is a conversion specification defines a set of matching input sequences, as described below for each conversion wide-character. A conversion specification is executed in the following steps:

Input white-space wide-characters (as specified by **iswspace(3C)**) are skipped, unless the conversion specification includes a `[]`, `c`, or `n` conversion character.

An item is read from the input, unless the conversion specification includes an `n` conversion wide-character. An input item is defined as the longest sequence

of input wide-characters, not exceeding any specified field width, which is an initial subsequence of a matching sequence. The first wide-character, if any, after the input item remains unread. If the length of the input item is 0, the execution of the conversion specification fails; this condition is a matching failure, unless end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is an input failure.

Except in the case of a % conversion wide-character, the input item (or, in the case of a % *n* conversion specification, the count of input wide-characters) is converted to a type appropriate to the conversion wide-character. If the input item is not a matching sequence, the execution of the conversion specification fails; this condition is a matching failure. Unless assignment suppression was indicated by a * , the result of the conversion is placed in the object pointed to by the first argument following the *format* argument that has not already received a conversion result if the conversion specification is introduced by % , or in the *n* th argument if introduced by the wide-character sequence % *n* \$. If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the space provided, the behavior is undefined.

The following conversion wide-characters are valid:

- d Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of `wcstol(3C)` with the value 10 for the *base* argument. In the absence of a size modifier, the corresponding argument must be a pointer to `int` .
- i Matches an optionally signed integer, whose format is the same as expected for the subject sequence of `wcstol(3C)` with 0 for the *base* argument. In the absence of a size modifier, the corresponding argument must be a pointer to `int` .
- o Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of `wcstoul(3C)` with the value 8 for the *base* argument. In the absence of a size modifier, the corresponding argument must be a pointer to `unsigned int` .
- u Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of `wcstoul(3C)` with the value 10 for the *base* argument. In the absence of a size modifier, the corresponding argument must be a pointer to `unsigned int` .
- x Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of `wcstoul(3C)` with the value 16 for the *base* argument. In the absence of a size modifier, the corresponding argument must be a pointer to `unsigned int` .

e , f , g Matches an optionally signed floating-point number, whose format is the same as expected for the subject sequence of `wcstod(3C)`. In the absence of a size modifier, the corresponding argument must be a pointer to `float`.

If the **fwprintf()** family of functions generates character string representations for infinity and NaN (a 7858 symbolic entity encoded in floating-point format) to support the ANSI/IEEE Std 754:1985 standard, the **fwscanf()** family of functions will recognize them as input.

s Matches a sequence of non white-space wide-characters. If no **l** (ell) qualifier is present, characters from the input field are converted as if by repeated calls to the `wcrtomb(3C)` function, with the conversion state described by an `mbstate_t` object initialized to zero before the first wide-character is converted. The corresponding argument must be a pointer to a character array large enough to accept the sequence and the terminating null character, which will be added automatically.

Otherwise, the corresponding argument must be a pointer to an array of `wchar_t` large enough to accept the sequence and the terminating null wide-character, which will be added automatically.

[Matches a non-empty sequence of wide-characters from a set of expected wide-characters (the *scanset*). If no **l** (ell) qualifier is present, wide-characters from the input field are converted as if by repeated calls to the `wcrtomb()` function, with the conversion state described by an `mbstate_t` object initialized to zero before the first wide-character is converted. The corresponding argument must be a pointer to a character array large enough to accept the sequence and the terminating null character, which will be added automatically.

If an **l** (ell) qualifier is present, the corresponding argument must be a pointer to an array of `wchar_t` large enough to accept the sequence and the terminating null wide-character, which will be added automatically.

The conversion specification includes all subsequent `widw` characters in the *format* string up to and including the matching right square bracket (`]`). The wide-characters between the square brackets (the *scanlist*) comprise the *scanset*, unless the wide-character after the left square bracket is a circumflex (`^`), in which case the *scanset* contains all wide-characters that do not appear in the *scanlist* between the circumflex and the right square bracket. If the conversion specification begins with `[]` or `[^]`, the right square bracket is included in the *scanlist* and the next right square bracket is the matching right square

bracket that ends the conversion specification; otherwise the first right square bracket is the one that ends the conversion specification. If a minus-sign (-) is in the scanlist and is not the first wide-character, nor the second where the first wide-character is a ^ , nor the last wide-character, it indicates a range of characters to be matched.

c Matches a sequence of wide-characters of the number specified by the field width (1 if no field width is present in the conversion specification). If no **l** (ell) qualifier is present, wide-characters from the input field are converted as if by repeated calls to the **wcrtomb()** function, with the conversion state described by an **mbstate_t** object initialized to zero before the first wide-character is converted. The corresponding argument must be a pointer to a character array large enough to accept the sequence. No null character is added.

Otherwise, the corresponding argument must be a pointer to an array of **wchar_t** large enough to accept the sequence. No null wide-character is added.

p Matches the set of sequences that is the same as the set of sequences that is produced by the **%p** conversion of the corresponding **fwprintf(3S)** functions. The corresponding argument must be a pointer to a pointer to **void** . If the input item is a value converted earlier during the same program execution, the pointer that results will compare equal to that value; otherwise the behavior of the **%p** conversion is undefined.

n No input is consumed. The corresponding argument must be a pointer to the integer into which is to be written the number of wide-characters read from the input so far by this call to the **fwscanf()** functions. Execution of a **%n** conversion specification does not increment the assignment count returned at the completion of execution of the function.

C Same as **lc** .

S Same as **ls** .

% Matches a single **%** ; no conversion or assignment occurs. The complete conversion specification must be **%%** .

If a conversion specification is invalid, the behavior is undefined.

The conversion characters **E** , **G** , and **X** are also valid and behave the same as, respectively, **e** , **g** , and **x** .

If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before any wide-characters matching the current conversion specification (except for %n) have been read (other than leading white-space, where permitted), execution of the current conversion specification terminates with an input failure. Otherwise, unless execution of the current conversion specification is terminated with a matching failure, execution of the following conversion specification (if any) is terminated with an input failure.

Reaching the end of the string in **swscanf()** is equivalent to encountering end-of-file for **fwscanf()**.

If conversion terminates on a conflicting input, the offending input is left unread in the input. Any trailing white space (including newline) is left unread unless matched by a conversion specification. The success of literal matches and suppressed assignments is only directly determinable via the %n conversion specification.

The **fwscanf()** and **wscanf()** functions may mark the `st_atime` field of the file associated with *stream* for update. The `st_atime` field will be marked for update by the first successful execution of **fgetc(3S)**, **fgetwc(3S)**, **fgets(3S)**, **fgetws(3S)**, **fread(3S)**, **getc(3S)**, **getwc(3S)**, **getchar(3S)**, **getwchar(3S)**, **gets(3S)**, **fscanf(3S)** or **fwscanf()** using *stream* that returns data not supplied by a prior call to **ungetc(3S)**.

RETURN VALUES

Upon successful completion, these functions return the number of successfully matched and assigned input items; this number can be 0 in the event of an early matching failure. If the input ends before the first matching failure or conversion, EOF is returned. If a read error occurs the error indicator for the stream is set, EOF is returned, and `errno` is set to indicate the error.

ERRORS

For the conditions under which the **fwscanf()** functions will fail and may fail, refer to **fgetwc(3S)**.

In addition, **fwscanf()** may fail if:

- EILSEQ** Input byte sequence does not form a valid character.
- EINVAL** There are insufficient arguments.

USAGE

In format strings containing the % form of conversion specifications, each argument in the argument list is used exactly once.

EXAMPLES

EXAMPLE 1 **wscanf()** example

The call:

```
int i, n; float x; char name[50];
n = wscanf(L"%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E-1 Hamster
```

will assign to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* will contain the string Hamster.

The call:

```
int i; float x; char name[50];
(void) wscanf(L"%2d%f*d %[0123456789]", &i, &x, name);
```

with input:

```
56789 0123 56a72
```

will assign 56 to *i*, 789.0 to *x*, skip 0123, and place the string 56\\0 in *name*. The next call to `getchar(3S)` will return the character a.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`fgetc(3S)`, `fgets(3S)`, `fgetwc(3S)`, `fgetws(3S)`, `fread(3S)`, `fscanf(3S)`, `fwprintf(3S)`, `getc(3S)`, `getchar(3S)`, `gets(3S)`, `getwc(3S)`, `getwchar(3S)`, `setlocale(3C)`, `wcrtomb(3C)`, `wcstod(3C)`, `wcstol(3C)`, `wcstoul(3C)`, `attributes(5)`, `standards(5)`

NAME | gelf, gelf_fsize, gelf_getclass, gelf_getdyn, gelf_getehdr, gelf_getphdr, gelf_getrel, gelf_getrela, gelf_getshdr, gelf_getsym, gelf_getsyminfo, gelf_newehdr, gelf_newphdr, gelf_update_dyn, gelf_update_ehdr, gelf_update_phdr, gelf_update_rel, gelf_update_rela, gelf_update_shdr, gelf_update_sym, gelf_update_syminfo, gelf_xlatetof, gelf_xlatetom – generic class-independent ELF interface

SYNOPSIS

```
cc
[flag ...]

file
...
-l
elf
[library ...]

#include <gelf.h>

int gelf_getclass(Elf *);

size_t gelf_fsize(Elf * elf, Elf_Type type, size_t cnt, unsigned ver);

GElf_Ehdr * gelf_getehdr(Elf * elf, GElf_Ehdr * dst);

int gelf_update_ehdr(Elf * elf, GElf_Ehdr * src);

unsigned long gelf_newehdr(Elf * elf, int class);

GElf_Phdr * gelf_getphdr(Elf * elf, int ndx, GElf_Phdr * src);

int gelf_update_phdr(Elf * elf, int ndx, GElf_Phdr * src);

unsigned long gelf_newphdr(Elf * elf, size_t phnum);

GElf_Shdr * gelf_getshdr(Elf_Scn * scn, Elf_Data * dst);

int gelf_update_shdr(Elf_Scn * scn, GElf_Shdr * src);

Elf_Data * gelf_xlatetof(Elf * elf, Elf_Data * dst, const Elf_Data * src, unsigned
encode);

Elf_Data * gelf_xlatetom(Elf * elf, Elf_Data * dst, const Elf_Data * src, unsigned
encode);

GElf_Sym * gelf_getsym(Elf_Data * data, int ndx, GElf_Sym * dst);

int gelf_update_sym(Elf_Data * dest, int ndx, GElf_Sym * src);
```

```

GElf_Dyn * gelf_getdyn(Elf_Data * src, int ndx, GElf_Dyn * src);
int gelf_update_dyn(Elf_Data * src, int ndx, GElf_Dyn * src);
GElf_Rela * gelf_getrela(Elf_Data * src, int ndx, GElf_Rela * dst);
int gelf_update_rela(Elf_Data * dst, int ndx, GElf_Rela * src);
GElf_Rel * gelf_getrel(Elf_Data * src, int ndx, GElf_Rel * dst);
int gelf_update_rel(Elf_Data * dst, int ndx, GElf_Rel * src);
GElf_Syminfo * gelf_getsyminfo(Elf_Data * src, int ndx, GElf_Syminfo * dst);
int gelf_update_syminfo(Elf_Data * dst, int ndx, GElf_Syminfo * src);
GElf_Move * gelf_getmove(Elf_Data * src, int ndx, GElf_Move * dst);
int gelf_update_move(Elf_Data * dst, int ndx, GElf_Move * src);

```

DESCRIPTION

`GElf` is a generic, ELF class-independent API, for manipulating ELF object files. `GElf` provides a single, common interface for handling 32-bit and 64-bit ELF format object files. `GElf` is a translation layer between the application and the class-dependent parts of the ELF library. Thus, the application can use `GElf`, which in turn, will call the corresponding `elf32_` or `elf64_` functions on behalf of the application. The data structures returned are all large enough to hold 32-bit and 64-bit data.

`GElf` provides a simple, class-independent layer of indirection over the class-dependent ELF32 and ELF64 API's. `GElf` is stateless, and may be used along side the ELF32 and ELF64 API's.

`GElf` always returns a copy of the underlying ELF32 or ELF64 structure, and therefore the programming practice of using the address of an ELF header as the base offset for the ELF's mapping into memory should be avoided. Also, data accessed by type-casting the `Elf_Data` buffer to a class-dependent type and treating it like an array, for example, a symbol table, will not work under `GElf`, and the `gelf_get` functions must be used instead. See the `EXAMPLE` section.

Programs which create or modify ELF files using `libelf(4)` need to perform an extra step when using `GElf`. Modifications to `GElf` values must be explicitly flushed to the underlying ELF32 or ELF64 structures by way of the `gelf_update_` interfaces. Use of `elf_update` or `elf_flagelf` and the like remains the same.

The sizes of versioning structures remains the same between ELF32 and ELF64. The `GElf` API only defines types for versioning, rather than a functional API. The processing of versioning information will stay the same in the `GElf` environment as it was in the class-dependent ELF environment.

List of Functions

| | |
|---------------------------|--|
| gelf_getclass() | Returns one of the constants <code>ELFCLASS32</code> , <code>ELFCLASS64</code> or <code>ELFCLASSNONE</code> . |
| gelf_fsize() | An analog to <code>elf32_fsize(3E)</code> and <code>elf64_fsize(3E)</code> . |
| gelf_getehdr() | An analog to <code>elf32_getehdr(3E)</code> and <code>elf64_getehdr(3E)</code> . |
| gelf_update_ehdr() | Copies the contents of the <code>GElf_Ehdr</code> ELF header to the underlying <code>Elf32_Ehdr</code> or <code>Elf64_Ehdr</code> structure. |
| gelf_newehdr() | An analog to <code>elf32_newehdr(3E)</code> and <code>elf64_newehdr(3E)</code> . |
| gelf_getphdr() | An analog to <code>elf32_getphdr(3E)</code> and <code>elf64_getphdr(3E)</code> . |
| gelf_update_phdr() | Copies of the contents of <code>GElf_Phdr</code> program header to underlying the <code>Elf32_Phdr</code> or <code>Elf64_Phdr</code> structure. |
| gelf_newphdr() | An analog to <code>elf32_newphdr(3E)</code> and <code>elf64_newphdr(3E)</code> . |
| gelf_getshdr() | An analog to <code>elf32_getshdr(3E)</code> and <code>elf64_getshdr(3E)</code> . |
| gelf_update_shdr() | Copies of the contents of <code>GElf_Shdr</code> section header to underlying the <code>Elf32_Shdr</code> or <code>Elf64_Shdr</code> structure. |
| gelf_xlateof() | An analog to <code>elf32_xlateof(3E)</code> and <code>elf64_xlateof(3E)</code> |
| gelf_xlatetom() | An analog to <code>elf32_xlatetom(3E)</code> and <code>elf64_xlatetom(3E)</code> |
| gelf_getsym() | Retrieves the <code>Elf32_Sym</code> or <code>Elf64_Sym</code> information from the symbol table at the given index. |
| gelf_update_sym() | Copies the <code>GElf_Sym</code> information back into the underlying <code>Elf32_Sym</code> or <code>Elf64_Sym</code> structure at the given index. |

| | |
|------------------------------|--|
| gelf_getdyn() | Retrieves the <code>Elf32_Dyn</code> or <code>Elf64_Dyn</code> information from the dynamic table at the given index. |
| gelf_update_dyn() | Copies the <code>GElf_Dyn</code> information back into the underlying <code>Elf32_Dyn</code> or <code>Elf64_Dyn</code> structure at the given index. |
| gelf_getrela() | Retrieves the <code>Elf32_Rela</code> or <code>Elf64_Rela</code> information from the relocation table at the given index. |
| gelf_update_rela() | Copies the <code>GElf_Rela</code> information back into the underlying <code>Elf32_Rela</code> or <code>Elf64_Rela</code> structure at the given index. |
| gelf_getrel() | Retrieves the <code>Elf32_Rel</code> or <code>Elf64_Rel</code> information from the relocation table at the given index. |
| gelf_update_rel() | Copies the <code>GElf_Rel</code> information back into the underlying <code>Elf32_Rel</code> or <code>Elf64_Rel</code> structure at the given index. |
| gelf_getsyminfo() | Retrieves the <code>Elf32_Syminfo</code> or <code>Elf64_Syminfo</code> information from the relocation table at the given index. |
| gelf_update_syminfo() | Copies the <code>GElf_Syminfo</code> information back into the underlying <code>Elf32_Syminfo</code> or <code>Elf64_Syminfo</code> structure at the given index. |
| gelf_getmove() | Retrieves the <code>Elf32_Move</code> or <code>Elf64_Move</code> information from the move table at the given index. |
| gelf_update_move() | Copies the <code>GElf_Move</code> information back into the underlying <code>Elf32_Move</code> or <code>Elf64_Move</code> structure at the given index. |

RETURN VALUES

Upon failure, all `GElf` functions return 0 and set `elf_errno`. See [elf_errno\(3E\)](#)

EXAMPLES**EXAMPLE 1** Printing the ELF Symbol Table

```
#include <stdio.h>
#include <sys/types.h>
```

```

#include <sys/stat.h>
#include <fcntl.h>
#include <libelf.h>
#include <gelf.h>

void
main(int argc, char **argv)
{
    Elf          *elf;
    Elf_Scn      *scn = NULL;
    GElf_Shdr    shdr;
    Elf_Data     *data;
    int          fd, ii, count;

    elf_version(EV_CURRENT);

    fd = open(argv[1], O_RDONLY);
    elf = elf_begin(fd, ELF_C_READ, NULL);

    while ((scn = elf_nextscn(elf, scn)) != NULL) {
        gelf_getshdr(scn, &shdr);
        if (shdr.sh_type == SHT_SYMTAB) {
            /* found a symbol table, go print it. */
            break;
        }
    }

    data          = elf_getdata(scn, NULL);
    count         = shdr.sh_size / shdr.sh_entsize;

    /* print the symbol names */
    for (ii=0; ii < count; ++ii) {
        GElf_Sym sym;
        gelf_getsym(data, ii, &sym);
        printf("%s\n", elf_strptr(elf, shdr.sh_link, sym.st_name));
    }
    elf_end(elf);
    close(fd);
}

```

FILES

| | |
|------------------------------|-----------------------|
| /usr/lib/libelf.so.1 | Shared object. |
| /usr/lib/sparcv9/libelf.so.1 | 64-bit shared object. |
| /usr/lib//libelf.a | Archive library. |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`elf(3E)`, `elf32_fsize(3E)`, `elf32_getehdr(3E)`, `elf32_newehdr(3E)`,
`elf32_getphdr(3E)`, `elf32_newphdr(3E)`, `elf32_getshdr(3E)`,
`elf32_xlateof(3E)`, `elf32_xlatetom(3E)`, `elf_errno(3E)`, `libelf(4)`,
`attributes(5)`

| | |
|--------------------|--|
| NAME | getacinfo, getacdir, getacflg, getacmin, getacna, setac, endac – get audit control file information |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library</i> ...] #include <bsm/libbsm.h> int getacdir(char * <i>dir</i>, int <i>len</i>); int getacmin(int * <i>min_val</i>); int getacflg(char * <i>auditstring</i>, int <i>len</i>); int getacna(char * <i>auditstring</i>, int <i>len</i>); void setac(void); void endac(void); </pre> |
| DESCRIPTION | <p>When first called, getacdir() provides information about the first audit directory in the <code>audit_control</code> file; thereafter, it returns the next directory in the file. Successive calls list all the directories listed in <code>audit_control(4)</code>. The parameter <i>len</i> specifies the length of the buffer <i>dir</i>. On return, <i>dir</i> points to the directory entry.</p> <p>getacmin() reads the minimum value from the <code>audit_control</code> file and returns the value in <i>min_val</i>. The minimum value specifies how full the file system to which the audit files are being written can get before the script <code>audit_warn(1M)</code> is invoked.</p> <p>getacflg() reads the system audit value from the <code>audit_control</code> file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>.</p> |

getacna() reads the system audit value for non-attributable audit events from the `audit_control` file and returns the value in *auditstring*. The parameter *len* specifies the length of the buffer *auditstring*. Non-attributable events are events that cannot be attributed to an individual user. `inetd(1M)` and several other daemons record non-attributable events.

Calling `setac` rewinds the `audit_control` file to allow repeated searches.

Calling `endac` closes the `audit_control` file when processing is complete.

FILES

`/etc/security/audit_control` contains default parameters read by the audit daemon, `auditd(1M)`

RETURN VALUES

getacdir(), **getacflg()**, **getacna()** and **getacmin()** return:

0 on success.

-2 on failure and set `errno` to indicate the error.

getacmin() and **getacflg()** return:

1 on EOF.

getacdir() returns:

-1 on EOF.

2 if the directory search had to start from the beginning because one of the other functions was called between calls to **getacdir()**.

These functions return:

-3 if the directory entry format in the `audit_control` file is incorrect.

getacdir(), **getacflg()** and **getacna()** return:

-3 if the input buffer is too short to accommodate the record.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe. |

SEE ALSO

`audit_warn(1M)`, `bsmconv(1M)`, `inetd(1M)`, `audit_control(4)`, `attributes(5)`

NOTES

The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See `bsmconv(1M)` for more information.

| | |
|--------------------|---|
| NAME | getauclassent, getauclassnam, setauclass, endauclass, getauclassnam_r, getauclassent_r – get audit_class entry |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lbsm -lsocket -lns1 -lintl [<i>library</i> ...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_class_ent * getauclassnam(const char * <i>name</i>); struct au_class_ent * getauclassnam_r(au_class_ent_t * <i>class_int</i>, const char * <i>name</i>); struct au_class_ent * getauclassent(void); struct au_class_ent * getauclassent_r(au_class_ent_t * <i>class_int</i>); void setauclass(void); void endauclass(void);</pre> |
| DESCRIPTION | <p>getauclassent() and getauclassnam() each return an audit_class entry.</p> <p>getauclassnam() searches for an audit_class entry with a given class name <i>name</i>.</p> <p>getauclassent() enumerates audit_class entries: successive calls to getauclassent() will return either successive audit_class entries or NULL.</p> <p>setauclass() “rewinds” to the beginning of the enumeration of audit_class entries. Calls to getauclassnam() may leave the enumeration in an indeterminate state, so setauclass() should be called before the first getauclassent() .</p> |

endauclass() may be called to indicate that `audit_class` processing is complete; the system may then close any open `audit_class` file, deallocate storage, and so forth.

getauclassent_r() and **getauclassnam_r()** both return a pointer to an `audit_class` entry as do their similarly named counterparts. They each take an additional argument, a pointer to pre-allocated space for an `au_class_ent_t`, which is returned if the call is successful. To assure there is enough space for the information returned, the applications programmer should be sure to allocate `AU_CLASS_NAME_MAX` and `AU_CLASS_DESC_MAX` bytes for the `ac_name` and `ac_desc` elements of the `au_class_ent_t` data structure.

The internal representation of an `audit_user` entry is an `au_class_ent` structure defined in `<bsm/libbsm.h>` with the following members:

```
char      *ac_name;
au_class_t\011ac_class;
char      *ac_desc;
```

RETURN VALUES

getauclassnam() and **getauclassnam_r()** return a pointer to a `struct au_class_ent` if they successfully locate the requested entry; otherwise they return `NULL`.

getauclassent() and **getauclassent_r()** return a pointer to a `struct au_class_ent` if they successfully enumerate an entry; otherwise they return `NULL`, indicating the end of the enumeration.

FILES

`/etc/security/audit_class` Maps audit class numbers to audit class names

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|--------------------------|
| MT-Level | MT-Safe with exceptions. |

All of the functions described in this man-page are MT-Safe except **getauclassent()** and `getauclassnam`. The two functions, **getauclassent_r()** and **getauclassnam_r()** have the same functionality as the unsafe functions, but have a slightly different function call interface in order to make them MT-Safe.

SEE ALSO

`bsmconv(1M)`, `audit_class(4)`, `audit_event(4)`, `attributes(5)`

NOTES

All information is contained in a static area, so it must be copied if it is to be saved.

The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See `bsmconv(1M)` for more information.

| | |
|----------------------|---|
| NAME | getauditflags, getauditflagsbin, getauditflagschar – convert audit flag specifications |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lbsm -lsocket -lns1 -lintl [<i>library</i> ...] #include <sys/param.h> #include <bsm/libbsm.h> int getauditflagsbin(char * <i>auditstring</i>, au_mask_t * <i>masks</i>); int getauditflagschar(char * <i>auditstring</i>, au_mask_t * <i>masks</i>, int <i>verbose</i>);</pre> |
| DESCRIPTION | <p>getauditflagsbin() converts the character representation of audit values pointed to by <i>auditstring</i> into <code>au_mask_t</code> fields pointed to by <i>masks</i> . These fields indicate which events are to be audited when they succeed and which are to be audited when they fail. The character string syntax is described in <code>audit_control(4)</code> .</p> <p>getauditflagschar() converts the <code>au_mask_t</code> fields pointed to by <i>masks</i> into a string pointed to by <i>auditstring</i> . If <i>verbose</i> is zero, the short (2-character) flag names are used. If <i>verbose</i> is non-zero, the long flag names are used. <i>auditstring</i> should be large enough to contain the ASCII representation of the events.</p> <p><i>auditstring</i> contains a series of event names, each one identifying a single audit class, separated by commas. The <code>au_mask_t</code> fields pointed to by <i>masks</i> correspond to binary values defined in <code><bsm/audit.h></code> , which is read by <code><bsm/libbsm.h></code> .</p> |
| RETURN VALUES | getauditflagsbin() and getauditflagschar() : -1 is returned on error and 0 on success. |
| ATTRIBUTES | See <code>attributes(5)</code> for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe. |

SEE ALSO `bsmconv(1M)`, `audit.log(4)`, `audit_control(4)`, `attributes(5)`

BUGS This is not a very extensible interface.

NOTES The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See `bsmconv(1M)` for more information.

| | |
|--------------------|---|
| NAME | getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent, getauevent_r, getauevnam_r, getauevnum_r – get audit_event entry |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lbsm -lsocket -lns1 -lintl [<i>library</i> ...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_event_ent * getauevent(void); struct au_event_ent * getauevnam(char * <i>name</i>); struct au_event_ent * getauevnum(au_event_t <i>event_number</i>); au_event_t * getauevnonam(char * <i>event_name</i>); void setauevent(void); void endauevent(void); struct au_event_ent * getauevent_r(au_event_ent_t * <i>e</i>); struct au_event_ent * getauevnam_r(au_event_ent_t * <i>e</i>, char * <i>name</i>); struct au_event_ent * getauevnum_r(au_event_ent_t * <i>e</i>, au_event_t <i>event_number</i>); </pre> |
| DESCRIPTION | <p>These interfaces document the programming interface for obtaining entries from the <code>audit_event(4)</code> file. <code>getauevent()</code>, <code>getauevnam()</code>, <code>getauevnum()</code>, <code>getauevent_r()</code>, <code>getauevnam_r()</code>, and <code>getauevnum_r()</code> each return a pointer to an <code>audit_event</code> structure.</p> <p><code>getauevent()</code> and <code>getauevent_r()</code> enumerate <code>audit_event</code> entries; successive calls to these functions will return either successive <code>audit_event</code> entries or <code>NULL</code>.</p> |

getauevnam() and **getauevnam_r()** search for an `audit_event` entry with a given `event_name` .

getauevnum() and **getauevnum_r()** search for an `audit_event` entry with a given `event_number` .

getauevnonam() searches for an `audit_event` entry with a given `event_name` and returns the corresponding event number.

setauevent() “rewinds” to the beginning of the enumeration of `audit_event` entries. Calls to **getauevnam()** , **getauevnum()** , **getauevnonam()** , **getauevnam_r()** , or **getauevnum_r()** may leave the enumeration in an indeterminate state; **setauevent()** should be called before the first **getauevent()** or **getauevent_r()** .

endauevent() may be called to indicate that `audit_event` processing is complete; the system may then close any open `audit_event` file , deallocate storage, and so forth.

The three functions **getauevent_r()** , **getauevnam_r()** , and **getauevnum_r()** each take an argument `e` which is a pointer to an `au_event_ent_t` . This pointer is returned on a successful function call. To assure there is enough space for the information returned, the applications programmer should be sure to allocate `AU_EVENT_NAME_MAX` and `AU_EVENT_DESC_MAX` bytes for the `ae_name` and `ac_desc` elements of the `au_event_ent_t` data structure.

The internal representation of an `audit_event` entry is an `struct au_event_ent` structure defined in `<bsm/libbsm.h>` with the following members:

```
au_event_t      ae_number
char            *ae_name;
char            *ae_desc*;
au_class_t     ae_class;
```

RETURN VALUES

getauevent() , **getauevnam()** , **getauevnum()** , **getauevent_r()** , **getauevnam_r()** , and **getauevnum_r()** return a pointer to a `struct au_event_ent` if the requested entry is successfully located; otherwise it returns `NULL` .

getauevnonam() returns an event number of type `au_event_t` if it successfully enumerates an entry; otherwise it returns `NULL` , indicating it could not find the requested event name.

FILES

| | |
|---------------------------|--|
| /etc/security/audit_event | Maps audit event numbers to audit event names. |
| /etc/passwd | Stores user-ID to username mappings. |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|--------------------------|
| MT-Level | MT-Safe with exceptions. |

The functions **getauevent()**, **getauevnam()**, and **getauevnum()** are not MT-Safe; however, there are equivalent functions: **getauevent_r()**, **getauevnam_r()**, and **getauevnum_r()** — all of which provide the same functionality and a MT-Safe function call interface.

SEE ALSO

bsmconv(1M), **getauclassent(3)**, **getpwnam(3C)**, **audit_class(4)**, **audit_event(4)**, **passwd(4)**, **attributes(5)**

NOTES

All information for the functions **getauevent()**, **getauevnam()**, and **getauevnum()** is contained in a static area, so it must be copied if it is to be saved.

The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See **bsmconv(1M)** for more information.

| | |
|--------------------|--|
| NAME | getauusernam, getauuserent, setauuser, endauuser – get audit_user entry |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lbsm -lsocket -lns1 -lintl [<i>library</i> ...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_user_ent * getauusernam(const char * <i>name</i>); struct au_user_ent * getauuserent(void); void setauuser(void); void endauuser(void); struct au_user_ent * getauusernam_r(au_user_ent_t * <i>u</i>, const char * <i>name</i>); struct au_user_ent * getauuserent_r(au_user_ent_t * <i>u</i>); </pre> |
| DESCRIPTION | <p>The getauuserent() , getauusernam() , getauuserent_r() , and getauusernam_r() functions each return an audit_user entry.</p> <p>The getauusernam() and getauusernam_r() functions search for an audit_user entry with a given login name <i>name</i> .</p> <p>The getauuserent() and getauuserent_r() functions enumerate audit_user entries; successive calls to these functions will return either successive audit_user entries or NULL .</p> <p>The setauuser() function “rewinds” to the beginning of the enumeration of audit_user entries. Calls to getauusernam() and getauusernam_r() may leave the enumeration in an indeterminate state, so setauuser() should be called before the first call to getauuserent() or getauuserent_r() .</p> |

The **endauser()** function may be called to indicate that `audit_user` processing is complete; the system may then close any open `audit_user` file, deallocate storage, and so forth.

The **getauserent_r()** and **getausernam_r()** functions both take an argument `u`, which is a pointer to an `au_user_ent`. This is the pointer that is returned on successful function calls.

The internal representation of an `audit_user` entry is an `au_user_ent` structure defined in `<bsm/libbsm.h>` with the following members:

```
char      *au_name;

au_mask_t au_always;
au_mask_t au_never;
```

RETURN VALUES

The **getausernam()** function returns a pointer to a `struct au_user_ent` if it successfully locates the requested entry; otherwise it returns `NULL`.

The **getauserent()** function returns a pointer to a `struct au_user_ent` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

USAGE

The functionality described in this manual page is available only if the Basic Security Module (BSM) has been enabled. See **bsmconv(1M)** for more information.

FILES

```
/etc/security/audit_user      stores per-user audit event mask
/etc/passwd                    stores user-id to username mappings
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|--------------------------|
| MT-Level | MT-Safe with exceptions. |

SEE ALSO

bsmconv(1M), **getpwnam(3C)**, **audit_user(4)**, **passwd(4)**, **attributes(5)**

NOTES

All information for the **getauserent()** and **getausernam()** functions is contained in a static area, so it must be copied if it is to be saved.

The **getauusernam()** and **getauuserent()** functions are not MT-safe. The **getauusernam_r()** and **getauuserent_r()** functions provide the same functionality with interfaces that are MT-Safe.

| | |
|----------------------|--|
| NAME | getbegyx, getmaxyx, getparyx, getyx – get cursor or window coordinates |
| SYNOPSIS | <pre>#include <curses.h> void getbegyx(WINDOW * win, int y, int x); void getmaxyx(WINDOW * win, int y, int x); void getparyx(WINDOW * win, int y, int x); void getyx(WINDOW * win, int y, int x);</pre> |
| PARAMETERS | <p>win Is a pointer to a window.</p> <p>y stores the <i>y</i> coordinate for the cursor or origin. The getmaxyx() macro uses it to store the number of rows in the window.</p> <p>x stores the <i>x</i> coordinate for the cursor or origin. The getmaxyx() macro uses it to store the number of columns in the window.</p> |
| DESCRIPTION | <p>The getyx() macro stores the current cursor position of the specified window in <i>x</i> and <i>y</i>.</p> <p>The getparyx() macro stores the <i>x</i> and <i>y</i> coordinates (relative to the parent window) of the specified window's origin (upper-left corner). If <i>win</i> does not point to a subwindow, <i>x</i> and <i>y</i> are set to -1.</p> <p>The getbegyx() macro stores the <i>x</i> and <i>y</i> coordinates of the specified window's origin (upper-left corner).</p> <p>The getmaxyx() macro stores the numbers of rows in the specified window in <i>y</i> and the number of columns in <i>x</i>.</p> |
| RETURN VALUES | These macros do not return a value. |
| ERRORS | None. |

| | |
|----------------------|--|
| NAME | getcchar – get a wide character string (with rendition) from a <code>cchar_t</code> |
| SYNOPSIS | <pre>#include <curses.h> int getcchar(const cchar_t *wval, wchar_t *wch, attr_t *attrs, short *color_pair, void *opt);</pre> |
| PARAMETERS | <p>wval Is a pointer to a <code>cchar_t</code> object.</p> <p>wch Is a pointer to an object where a wide character string can be stored.</p> <p>attrs Is a pointer to an object where attributes can be stored.</p> <p>color_pair Is a pointer to an object where a color pair can be stored.</p> <p>opts Is reserved for future use. Currently, this must be a null pointer.</p> |
| DESCRIPTION | <p>If <i>wch</i> is not a null pointer, the getcchar() function splits the <code>cchar_t</code> object pointed to by <i>wval</i> into a wide character string, attributes, and a color pair. It stores the attributes in the location pointed to by <i>attrs</i>, the color pair in the location pointed to by <i>color_pair</i>, and the wide character string in the location pointed to by <i>wch</i>.</p> <p>If <i>wch</i> is a null pointer, the getcchar() function simply returns the number of wide characters in the <code>cchar_t</code> object pointed to by <i>wval</i>. The objects pointed to by <i>attrs</i> and <i>color_pair</i> are not changed.</p> |
| RETURN VALUES | <p>When <i>wch</i> is a null pointer, the getcchar() function returns the number of wide characters in the string pointed to by <i>wval</i> including the null terminator.</p> <p>When <i>wch</i> is not a null pointer, the getcchar() function returns <code>OK</code> on success and <code>ERR</code> otherwise.</p> |
| ERRORS | None |
| SEE ALSO | attroff(3XC) , can_change_color(3XC) , setcchar(3XC) |

| | |
|--------------------|--|
| NAME | getch, wgetch, mvwgetch, mvwgetch – get a single-byte character from terminal |
| SYNOPSIS | <pre>#include <curses.h> int getch(void); int wgetch(WINDOW * win); int mvwgetch(int y, int x); int mvwgetch(WINDOW * win, int y, int x);</pre> |
| PARAMETERS | <p>win Is a pointer to the window associated with the terminal from which the character is to be read.</p> <p>y Is the y (row) coordinate for the position of the character to be read.</p> <p>x Is the x (column) coordinate for the position of the character to be read.</p> |
| DESCRIPTION | <p>The getch() and wgetch() functions get a single-byte character from the terminal associated with the window <code>stdscr</code> or window <code>win</code>, respectively. The mvwgetch() and mvwgetch() functions move the cursor to the position specified in <code>stdscr</code> or <code>win</code>, respectively, then get a character.</p> <p>If the window is not a pad and has been changed since the last call to refresh(3XC), getch() calls refresh() to update the window before the next character is read.</p> <p>The setting of certain functions affects the behavior of the getch() set of functions. For example, if cbreak(3XC) is set, characters typed by the user are immediately processed. If halfdelay(3XC) is set, getch() waits until a character is typed or returns <code>ERR</code> if no character is typed within the specified timeout period. This timeout can also be specified for individual windows with the <i>delay</i> parameter of timeout(3XC). A negative value waits for input; a value of 0 returns <code>ERR</code> if no input is ready; a positive value blocks until input arrives or the time specified expires (in which case, <code>ERR</code> is returned). If nodelay(3XC) is set, <code>ERR</code> is returned if no input is waiting; if not set, getch() waits until input arrives. Each character will be echoed to the window unless noecho(3XC) has been set.</p> <p>If keypad handling is enabled (keypad(3XC) is <code>TRUE</code>), the token for the function key is returned. If a character is received that could be the beginning of a function key (for example, <code>ESC</code>), an inter-byte timer is set. If the remainder of the sequence is not received before the time expires, the character is passed through; otherwise, the value of the function key is returned. If notimeout() is set, the inter-byte timer is not used.</p> |

The ESC key is typically a prefix key used with function keys and should not be used as a single character.

The following is a list of tokens for function keys that are returned by the **getch()** set of functions if keypad handling is enabled (some terminals may not support all tokens).

Constant Values for Function Keys

| Constant | Description |
|-----------------|---------------------------------------|
| Y_BREAK | Break key |
| Y_DOWN | The down arrow key |
| Y_UP | The up arrow key |
| Y_LEFT | The left arrow key |
| Y_RIGHT | The right arrow key |
| Y_HOME | Home key |
| Y_BACKSPACE | Backspace |
| Y_F0 | Function keys. Space for 64 |
| Y_F(<i>n</i>) | (KEY_F0+(<i>n</i>)) key is reserved |
| Y_DL | Delete line |
| Y_IL | Insert line |
| Y_DC | Delete character |
| Y_IC | Insert char or enter insert mode |
| Y_EIC | Exit insert char mode |
| Y_CLEAR | Clear screen |
| Y_EOS | Clear to end of screen |
| Y_EOL | Clear to end of line |
| Y_SF | Scroll 1 line forward |
| Y_SR | Scroll 1 line backwards |
| Y_NPAGE | Next page |
| Y_PPAGE | Previous page |
| Y_STAB | Set tab |
| Y_CTAB | Clear tab |
| Y_CATAB | Clear all tabs |
| Y_ENTER | Enter or send |

| Constant | Description |
|-------------|----------------------------------|
| Y_SRESET | Soft (partial) reset |
| Y_RESET | Reset or hard reset |
| Y_PRINT | Print or copy |
| Y_LL | Home down or bottom (lower left) |
| Y_A1 | Upper left of keypad |
| Y_A3 | Upper right of keypad |
| Y_B2 | Center of keypad |
| Y_C1 | Lower left of keypad |
| Y_C3 | Lower right of keypad |
| Y_BTAB | Back tab |
| Y_BEG | Beginning key |
| Y_CANCEL | Cancel key |
| Y_CLOSE | Close key |
| Y_COMMAND | Cmd (command) key |
| Y_COPY | Copy key |
| Y_CREATE | Create key |
| Y_END | End key |
| Y_EXIT | Exit key |
| Y_FIND | Find key |
| Y_HELP | Help key |
| Y_MARK | Mark key |
| Y_MESSAGE | Message key |
| Y_MOVE | Move key |
| Y_NEXT | Next object key |
| Y_OPEN | Open key |
| Y_OPTIONS | Options key |
| Y_PREVIOUS | Previous object key |
| Y_REDO | Redo key |
| Y_REFERENCE | Ref(erence) key |
| Y_REFRESH | Refresh key |

| Constant | Description |
|-------------|-------------------------|
| Y_REPLACE | Replace key |
| Y_RESTART | Restart key |
| Y_RESUME | Resume key |
| Y_SAVE | Save key |
| Y_SBEG | Shifted beginning key |
| Y_SCANCEL | Shifted cancel key |
| Y_SCOMMAND | Shifted command key |
| Y_SCOPY | Shifted copy key |
| Y_SCREATE | Shifted create key |
| Y_SDC | Shifted delete char key |
| Y_SDL | Shifted delete line key |
| Y_SELECT | Select key |
| Y_SEND | Shifted end key |
| Y_SEOL | Shifted clear line key |
| Y_SEXIT | Shifted exit key |
| Y_SFIND | Shifted find key |
| Y_SHELP | Shifted help key |
| Y_SHOME | Shifted home key |
| Y_SIC | Shifted input key |
| Y_SLEFT | Shifted left key |
| Y_SMESSAGES | Shifted messages key |
| Y_SMOVE | Shifted move key |
| Y_SNEXT | Shifted next key |
| Y_SOPTIONS | Shifted options key |
| Y_SPREVIOUS | Shifted previous key |
| Y_SPRINT | Shifted print key |
| Y_SREDO | Shifted redo key |
| Y_SREPLACE | Shifted replace key |
| Y_SRIGHT | Shifted right key |
| Y_SRSUME | Shifted resume key |

| Constant | Description |
|------------|---------------------|
| Y_SSAVE | Shifted save key |
| Y_SSUSPEND | Shifted suspend key |
| Y_SUNDO | Shifted undo key |
| Y_SUSPEND | Suspend key |
| Y_UNDO | Undo key |

RETURN VALUES

On success, these function return `OK` . Otherwise, they return `ERR` .

ERRORS

None

SEE ALSO

`cbreak(3XC)` , `echo(3XC)` , `halfdelay(3XC)` , `keypad(3XC)` ,
`nodelay(3XC)` , `notimeout(3XC)` , `raw(3XC)` , `timeout(3XC)`

| | |
|----------------------|---|
| NAME | getcwd – get pathname of current working directory |
| SYNOPSIS | <pre>#include <unistd.h> extern char *getcwd(char *buf, size_t size);</pre> |
| DESCRIPTION | <p>The getcwd() function returns a pointer to the current directory pathname. The value of <i>size</i> must be at least one greater than the length of the pathname to be returned.</p> <p>If <i>buf</i> is not <code>NULL</code>, the pathname will be stored in the space pointed to by <i>buf</i>.</p> <p>If <i>buf</i> is a null pointer, getcwd() will obtain <i>size</i> bytes of space using malloc(3C). In this case, the pointer returned by getcwd() may be used as the argument in a subsequent call to free().</p> |
| RETURN VALUES | The getcwd() function returns <code>NULL</code> with <code>errno</code> set if <i>size</i> is not large enough, or if an error occurs in a lower-level function. |
| ERRORS | <p>The getcwd() function will fail if:</p> <p>EACCES A parent directory cannot be read to get its name.</p> <p>EINVAL The <i>size</i> argument is equal to 0.</p> <p>ERANGE The <i>size</i> argument is greater than 0 and less than the length of the pathname plus 1.</p> |
| USAGE | Applications should exercise care when using chdir(2) in conjunction with getcwd() . The current working directory is global to all threads within a process. If more than one thread calls chdir() to change the working directory, a subsequent call to getcwd() could produce results that are unexpected. |
| EXAMPLES | <p>EXAMPLE 1 Printing the current working directory.</p> <p>The following example prints the current working directory.</p> <pre>#include <unistd.h> #include <stdio.h> main() { char *cwd; if ((cwd = getcwd(NULL, 64)) == NULL) { perror("pwd"); exit(2); } (void)printf("%s\n", cwd); return(0); }</pre> |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`chdir(2)`, `malloc(3C)`, `attributes(5)`

| | |
|----------------------------------|--|
| NAME | getdate – convert user format date and time |
| SYNOPSIS | <pre>#include <time.h> struct tm *getdate(const char *string); extern int getdate_err;</pre> |
| DESCRIPTION | <p>The getdate() function converts user-definable date and/or time specifications pointed to by <i>string</i> to a <code>tm</code> structure. The <code>tm</code> structure is defined in the <code><time.h></code> header.</p> <p>User-supplied templates are used to parse and interpret the input string. The templates are text files created by the user and identified via the environment variable <code>DATEMSK</code>. Each line in the template represents an acceptable date and/or time specification using conversion specifications similar to those used by strftime(3C) and strptime(3C). Dates before 1902 and after 2037 are illegal. The first line in the template that matches the input specification is used for interpretation and conversion into the internal time format.</p> |
| Conversion Specifications | <p>The following conversion specifications are supported:</p> <ul style="list-style-type: none"> <code>%%</code> Same as <code>%</code>. <code>%a</code> Locale's abbreviated weekday name. <code>%A</code> Locale's full weekday name. <code>%b</code> Locale's abbreviated month name. <code>%B</code> Locale's full month name. <code>%c</code> Locale's appropriate date and time representation. <code>%C</code> Century number [0,99]; leading zero is permitted but not required. If used without the <code>%Y</code> specifier, this format specifier will assume the current year offset in whichever century is specified. The only valid years are between 1902-2037. <code>%d</code> day of month [01,31]; leading zero is permitted but not required. <code>%D</code> Date as <code>%m/%d/%y</code>. <code>%e</code> Same as <code>%d</code>. <code>%h</code> Locale's abbreviated month name. <code>%H</code> Hour (24-hour clock) [0,23]; leading zero is permitted but not required. <code>%I</code> Hour (12-hour clock) [1,12]; leading zero is permitted but not required. |

| | |
|----|--|
| %j | Day number of the year [1,366]; leading zeros are permitted but not required. |
| %m | Month number [1,12]; leading zero is permitted but not required. |
| %M | Minute [0,59]; leading zero is permitted but not required. |
| %n | Any white space. |
| %p | Locale's equivalent of either a.m. or p.m. |
| %r | Appropriate time representation in the 12-hour clock format with %p. |
| %R | Time as %H:%M. |
| %S | Seconds [0,61]; leading zero is permitted but not required. The range of values is [00,61] rather than [00,59] to allow for the occasional leap second and even more occasional double leap second. |
| %t | Any white space. |
| %T | Time as %H:%M:%S. |
| %U | Week number of the year as a decimal number [0,53], with Sunday as the first day of the week; leading zero is permitted but not required. |
| %w | Weekday as a decimal number [0,6], with 0 representing Sunday. |
| %W | Week number of the year as a decimal number [0,53], with Monday as the first day of the week; leading zero is permitted but not required. |
| %x | Locale's appropriate date representation. |
| %X | Locale's appropriate time representation. |
| %Y | Year within century. When a century is not otherwise specified, values in the range 69-99 refer to years in the twentieth century (1969 to 1999 inclusive); values in the range 00-68 refer to years in the twenty-first century (2000 to 2068 inclusive). |
| %y | Year, including the century (for example, 1993). |
| %Z | Time zone name or no characters if no time zone exists. |

Modified Conversion Specifications

Some conversion specifications can be modified by the E and O modifier characters to indicate that an alternative format or specification should be used

rather than the one normally used by the unmodified specification. If the alternative format or specification does not exist in the current locale, the behavior be as if the unmodified conversion specification were used.

- `%Ec` Locale's alternative appropriate date and time representation.
- `%EC` Name of the base year (period) in the locale's alternative representation.
- `%Ex` Locale's alternative date representation.
- `%EX` Locale's alternative time representation.
- `%EY` Offset from `%EC` (year only) in the locale's alternative representation.
- `%EY` Full alternative year representation.
- `%Od` Day of the month using the locale's alternative numeric symbols; leading zeros are permitted but not required.
- `%Oe` Same as `%Od`.
- `%OH` Hour (24-hour clock) using the locale's alternative numeric symbols.
- `%OI` Hour (12-hour clock) using the locale's alternative numeric symbols.
- `%Om` Month using the locale's alternative numeric symbols.
- `%OM` Minutes using the locale's alternative numeric symbols.
- `%OS` Seconds using the locale's alternative numeric symbols.
- `%OU` Week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.
- `%Ow` Number of the weekday (Sunday=0) using the locale's alternative numeric symbols.
- `%OW` Week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.
- `%Oy` Year (offset from `%C`) in the locale's alternative representation and using the locale's alternative numeric symbols.

**Internal Format
Conversion**

The following rules are applied for converting the input specification into the internal format:

- If only the weekday is given, today is assumed if the given day is equal to the current day and next week if it is less.
- If only the month is given, the current month is assumed if the given month is equal to the current month and next year if it is less and no year is given. (The first day of month is assumed if no day is given.)
- If the century is given, but the year within the century is not given, the current year within the century is assumed.
- If no hour, minute, and second are given, the current hour, minute, and second are assumed.
- If no date is given, today is assumed if the given hour is greater than the current hour and tomorrow is assumed if it is less.

General Specifications

A conversion specification that is an ordinary character is executed by scanning the next character from the buffer. If the character scanned from the buffer differs from the one comprising the conversion specification, the specification fails, and the differing and subsequent characters remain unscanned.

A series of conversion specifications composed of %n, %t, white space characters, or any combination is executed by scanning up to the first character that is not white space (which remains unscanned), or until no more characters can be scanned.

Any other conversion specification is executed by scanning characters until a character matching the next conversion specification is scanned, or until no more characters can be scanned. These characters, except the one matching the next conversion specification, are then compared to the locale values associated with the conversion specifier. If a match is found, values for the appropriate *tm* structure members are set to values corresponding to the locale information. If no match is found, **getdate()** fails and no more characters are scanned.

The month names, weekday names, era names, and alternative numeric symbols can consist of any combination of upper and lower case letters. The user can request that the input date or time specification be in a specific language by setting the LC_TIME category using **setlocale(3C)**.

RETURN VALUES

If successful, **getdate()** returns a pointer to a *tm* structure; otherwise, it returns `NULL` and sets the global variable `getdate_err` to indicate the error. Subsequent calls to **getdate()** alter the contents of `getdate_err`.

The following is a complete list of the `getdate_err` settings and their meanings:

- 1 The DATEMSK environment variable is null or undefined.
- 2 The template file cannot be opened for reading.

- 3 Failed to get file status information.
- 4 The template file is not a regular file.
- 5 An error is encountered while reading the template file.
- 6 The **malloc()** function failed (not enough memory is available).
- 7 There is no line in the template that matches the input.
- 8 The input specification is invalid (for example, February 31).

USAGE

The **getdate()** function makes explicit use of macros described on the **ctype(3C)** manual page.

EXAMPLES

EXAMPLE 1 Examples of the **getdate()** function.

The following example shows the possible contents of a template:

```
%m
%A %B %d %Y, %H:%M:%S
%A
%B
%m/%d/%y %I %p
%d,%m,%Y %H:%M
at %A the %dst of %B in %Y
run job at %I %p,%B %dnd
%A den %d. %B %Y %H.%M Uhr
```

The following are examples of valid input specifications for the above template:

```
getdate("10/1/87 4 PM")
getdate("Friday")
getdate("Friday September 19 1987, 10:30:30")
getdate("24,9,1986 10:30")
getdate("at monday the 1st of december in 1986")
getdate("run job at 3 PM, december 2nd")
```

If the **LANG** environment variable is set to **de** (German), the following is valid:

```
getdate("freitag den 10. oktober 1986 10.30 Uhr")
```

Local time and date specification are also supported. The following examples show how local date and time specification can be defined in the template.

| Invocation | Line in Template |
|----------------------------|------------------|
| getdate("11/27/86") | %m/%d/%y |
| getdate("27.11.86") | %d.%m.%y |
| getdate("86-11-27") | %y-%m-%d |
| getdate("Friday 12:00:00") | %A %H:%M:%S |

The following examples illustrate the Internal Format Conversion rules. Assume that the current date is Mon Sep 22 12:19:47 EDT 1986 and the LANG environment variable is not set.

| Input | Line in Template | Date |
|--------------|------------------|------------------------------|
| Mon | %a | Mon Sep 22 12:19:48 EDT 1986 |
| Sun | %a | Sun Sep 28 12:19:49 EDT 1986 |
| Fri | %a | Fri Sep 26 12:19:49 EDT 1986 |
| September | %B | Mon Sep 1 12:19:49 EDT 1986 |
| January | %B | Thu Jan 1 12:19:49 EST 1987 |
| December | %B | Mon Dec 1 12:19:49 EST 1986 |
| Sep Mon | %b %a | Mon Sep 1 12:19:50 EDT 1986 |
| Jan Fri | %b %a | Fri Jan 2 12:19:50 EST 1987 |
| Dec Mon | %b %a | Mon Dec 1 12:19:50 EST 1986 |
| Jan Wed 1989 | %b %a %Y | Wed Jan 4 12:19:51 EST 1989 |
| Fri 9 | %a %H | Fri Sep 26 09:00:00 EDT 1986 |
| Feb 10:30 | %b %H:%S | Sun Feb 1 10:00:30 EST 1987 |
| 10:30 | %H:%M | Tue Sep 23 10:30:00 EDT 1986 |
| 13:30 | %H:%M | Mon Sep 22 13:30:00 EDT 1986 |

ATTRIBUTES

See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |
| CSI | Enabled |

SEE ALSO | `ctype(3C)`, `setlocale(3C)`, `strftime(3C)`, `strptime(3C)`,
`attributes(5)`, `environ(5)`

| | |
|----------------------|---|
| NAME | getdtablesize – get the file descriptor table size |
| SYNOPSIS | <pre>#include <unistd.h> int getdtablesize(void);</pre> |
| DESCRIPTION | The getdtablesize() function is equivalent to getrlimit(2) with the RLIMIT_NOFILE option. |
| RETURN VALUES | The getdtablesize() function returns the current soft limit as if obtained from a call to getrlimit() with the RLIMIT_NOFILE option. |
| ERRORS | No errors are defined. |
| USAGE | <p>There is no direct relationship between the value returned by getdtablesize() and OPEN_MAX defined in <limits.h>.</p> <p>Each process has a file descriptor table which is guaranteed to have at least 20 slots. The entries in the descriptor table are numbered with small integers starting at 0. The getdtablesize() function returns the current maximum size of this table by calling the getrlimit() function.</p> |
| SEE ALSO | close(2) , getrlimit(2) , open(2) , setrlimit(2) , select(3C) |

| NAME | getenv – return value for environment name | | | | |
|----------------------|---|----------------|-----------------|----------|------|
| SYNOPSIS | #include <stdlib.h> char * getenv (const char * <i>name</i>); | | | | |
| DESCRIPTION | The getenv() function searches the environment list (see environ(5)) for a string of the form <i>name=value</i> and, if the string is present, returns a pointer to the <i>value</i> in the current environment. | | | | |
| RETURN VALUES | If successful, getenv() returns a pointer to the <i>value</i> in the current environment; otherwise, it returns a null pointer. | | | | |
| USAGE | The getenv() function can be safely called from a multithreaded application. Care must be exercised when using both getenv() and putenv(3C) in a multithreaded application. These functions examine and modify the environment list, which is shared by all threads in an application. The system prevents the list from being accessed simultaneously by two different threads. It does not, however, prevent two threads from successively accessing the environment list using getenv() or putenv(3C) . | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe | | | | |
| SEE ALSO | exec(2) , putenv(3C) , attributes(5) , environ(5) | | | | |

| NAME | getexecname – return pathname of executable | | | | |
|----------------------|---|----------------|-----------------|----------|------|
| SYNOPSIS | <pre>#include <stdlib.h> const char *getexecname(void);</pre> | | | | |
| DESCRIPTION | <p>The getexecname() function returns the pathname (the first argument of one of the exec family of functions; see exec(2)) of the executable that started the process.</p> <p>Normally this is an absolute pathname, as the majority of commands are executed by the shells that append the command name to the user's PATH components. If this is not an absolute path, the output of getcwd(3C) can be prepended to it to create an absolute path, unless the process or one of its ancestors has changed its root directory or current working directory since the last successful call to one of the exec family of functions.</p> | | | | |
| RETURN VALUES | If successful, getexecname() returns a pointer to the executables pathname; otherwise, it returns 0. | | | | |
| USAGE | <p>The getexecname() function obtains the executable pathname from the AT_SUN_EXECNAME aux vector. These vectors are made available to dynamically linked processes only.</p> <p>A successful call to one of the exec family of functions will always have AT_SUN_EXECNAME in the aux vector. The associated pathname is guaranteed to be less than or equal to PATH_MAX, not counting the trailing null byte that is always present.</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe | | | | |
| SEE ALSO | exec(2) , getcwd(3C) , attributes(5) | | | | |

NAME | getfauditflags – generates the process audit state

SYNOPSIS |

```
cc [ flag ... ] file ... -lbsm -lsocket -lnsl -lintl [ library ... ]
#include <sys/param.h>
#include <bsm/libbsm.h>
```

```
int getfauditflags(au_mask_t *usremasks, au_mask_t *usrdmasks, au_mask_t
*lastmasks);
```

DESCRIPTION | **getfauditflags()** generates a process audit state by combining the audit masks passed as parameters with the system audit masks specified in the **audit_control(4)** file. **getfauditflags()** obtains the system audit value by calling **getacflg()** (see **getacinfo(3)**).

usremasks points to **au_mask_t** fields which contains two values. The first value defines which events are *always* to be audited when they succeed. The second value defines which events are always to be audited when they fail.

usrdmasks also points to **au_mask_t** fields which contains two values. The first value defines which events are *never* to be audited when they succeed. The second value defines which events are never to be audited when they fail.

The structures pointed to by *usremasks* and *usrdmasks* may be obtained from the **audit_user(4)** file by calling **getauusernam()** which returns a pointer to a structure containing all **audit_user(4)** fields for a user.

The output of this function is stored in *lastmasks* which is a pointer of type **au_mask_t** as well. The first value defines which events are to be audited when they succeed and the second defines which events are to be audited when they fail.

Both *usremasks* and *usrdmasks* override the values in the system audit values.

RETURN VALUES | -1 is returned on error and 0 on success.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe. |

SEE ALSO | **bsmconv(1M)**, **getacinfo(3)**, **getauditflags(3)**, **getauusernam(3)**, **audit.log(4)**, **audit_control(4)**, **audit_user(4)**, **attributes(5)**

NOTES

The functionality described in this man page is available only if the Basic Security Module (BSM) has been enabled. See `bsmconv(1M)` for more information.

| | |
|--------------------|---|
| NAME | getgrnam, getgrnam_r, getgrent, getgrent_r, getgrgid, getgrgid_r, setgrent, endgrent, fgetgrent, fgetgrent_r – get group entry |
| SYNOPSIS | <pre>#include <grp.h> struct group * getgrnam(const char * <i>name</i>); struct group * getgrnam_r(const char * <i>name</i>, struct group * <i>grp</i>, char * <i>buffer</i>, int <i>buflen</i>); struct group * getgrent(void); struct group * getgrent_r(struct group * <i>grp</i>, char * <i>buffer</i>, int <i>buflen</i>); struct group * getgrgid(gid_t <i>gid</i>); struct group * getgrgid_r(gid_t <i>gid</i>, struct group * <i>grp</i>, char * <i>buffer</i>, int <i>buflen</i>); void setgrent(void); void endgrent(void); struct group * fgetgrent(FILE * <i>f</i>); struct group * fgetgrent_r(FILE * <i>f</i>, struct group * <i>grp</i>, char * <i>buffer</i>, int <i>buflen</i>);</pre> |
| POSIX | <pre>cc [<i>flag</i> ...] <i>file</i> ... -D_POSIX_PTHREAD_SEMANTICS [<i>library</i> ...] int getgrnam_r(const char * <i>name</i>, struct group * <i>grp</i>, char * <i>buffer</i>, size_t <i>bufsize</i>, struct group ** <i>result</i>); int getgrgid_r(gid_t <i>gid</i>, struct group * <i>grp</i>, char * <i>buffer</i>, size_t <i>bufsize</i>, struct group ** <i>result</i>);</pre> |
| DESCRIPTION | <p>These functions are used to obtain entries describing user groups. Entries can come from any of the sources for <code>group</code> specified in the <code>/etc/nsswitch.conf</code> file (see <code>nsswitch.conf(4)</code>).</p> <p>The <code>getgrnam()</code> function searches for an entry with the group name specified by the character string parameter <code>name</code>.</p> <p>The <code>getgrgid()</code> function searches for an entry with the (numeric) group id specified by <code>gid</code>.</p> |

The **setgrent()** , **getgrent()** , and **endgrent()** functions are used to enumerate group entries from the database. The **setgrent()** function sets (or resets) the enumeration to the beginning of the set of group entries. This function should be called before the first call to **getgrent()** . Calls to **getgrnam()** and **getgrgid()** leave the enumeration position in an indeterminate state. Successive calls to **getgrent()** return either successive entries or `NULL` , indicating the end of the enumeration.

The **endgrent()** function may be called to indicate that the caller expects to do no further group entry retrieval operations; the system may then close the group file, deallocate resources it was using, and so forth. It is still allowed, but possibly less efficient, for the process to call more group functions after calling **endgrent()** .

The **fgetgrent()** function, unlike the other functions above, does not use `nsswitch.conf` ; it reads and parses the next line from the stream *f* , which is assumed to have the format of the group file (see **group(4)**).

Reentrant Interfaces

The **getgrnam()** , **getgrgid()** , **getgrent()** , and **fgetgrent()** functions use static storage that is re-used in each call, making them unsafe for multithreaded applications.

The parallel functions **getgrnam_r()** , **getgrgid_r()** , **getgrent_r()** , and **fgetgrent_r()** provide reentrant interfaces for these operations.

Each reentrant interface performs the same operation as its non-reentrant counterpart, named by removing the `_r` suffix. The reentrant interfaces, however, use buffers supplied by the caller to store returned results, and are safe for use in both single-threaded and multithreaded applications.

Each reentrant interface takes the same parameters as its non-reentrant counterpart, as well as the following additional parameters. The *grp* parameter must be a pointer to a `struct group` structure allocated by the caller. On successful completion, the function returns the group entry in this structure. The parameter *buffer* is a pointer to a buffer supplied by the caller, used as storage space for the group data. All of the pointers within the returned `struct group` *grp* point to data stored within this buffer; see **RETURN VALUES** . The buffer must be large enough to hold all the data associated with the group entry. The parameter *buflen* (or *bufsize* for the POSIX versions; see **standards(5)**) should give the size in bytes of *buffer* . The POSIX versions place a pointer to the modified *grp* structure in the *result* parameter, instead of returning a pointer to this structure.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. **setgrent()** may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to **getgrent_r()** , the threads will

enumerate disjoint subsets of the group database. Like their non-reentrant counterparts, **getgrnam_r()** and **getgrgid_r()** leave the enumeration position in an indeterminate state.

RETURN VALUES

Group entries are represented by the `struct group` structure defined in `<grp.h>` :

```
struct group {
    char *gr_name;           /* the name of the group */
    char *gr_passwd;        /* the encrypted group password */
    gid_t gr_gid;           /* the numerical group ID */
    char **gr_mem;          /* vector of pointers to member names */
};
```

The **getgrnam()**, **getgrnam_r()**, **getgrgid()**, and **getgrgid_r()** functions each return a pointer to a `struct group` if they successfully locate the requested entry; otherwise they return `NULL`. The POSIX functions **getgrnam_r()** and **getgrgid_r()** return 0 upon success or the error number in case of failure.

The **getgrent()**, **getgrent_r()**, **fgetgrent()**, and **fgetgrent_r()** functions each return a pointer to a `struct group` if they successfully enumerate an entry; otherwise they return `NULL`, indicating the end of the enumeration.

The **getgrnam()**, **getgrgid()**, **getgrent()**, and **fgetgrent()** functions use static storage, so returned data must be copied before a subsequent call to any of these functions if the data is to be saved.

When the pointer returned by the reentrant functions **getgrnam_r()**, **getgrgid_r()**, **getgrent_r()**, and **fgetgrent_r()** is non-null, it is always equal to the `grp` pointer that was supplied by the caller.

ERRORS

The reentrant functions **getgrnam_r()**, **getgrgid_r()**, **getgrent_r()**, and **fgetgrent_r()** return `NULL` and set `errno` to `ERANGE` (or in the case of POSIX functions **getgrnam_r()** and **getgrgid_r()** return the `ERANGE` error) if the length of the buffer supplied by caller is not large enough to store the result. See **Intro(2)** for the proper usage and interpretation of `errno` in multithreaded applications.

FILES

```
/etc/group
/etc/nsswitch.conf
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|---|
| MT-Level | See "Reentrant Interfaces" in DESCRIPTION . |

SEE ALSO `getpwnam(3C)` , `group(4)` , `nsswitch.conf(4)` , `passwd(4)` , `attributes(5)` , `standards(5)`

NOTES When compiling multithreaded programs, see `Intro(3)` , *Notes On Multithreaded Applications* .

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

Use of the enumeration interfaces `getgrent()` and `getgrent_r()` is discouraged; enumeration is supported for the group file, NIS, and NIS+, but in general is not efficient and may not be supported for all database sources. The semantics of enumeration are discussed further in `nsswitch.conf(4)` .

Previous releases allowed the use of "+" and "-" entries in `/etc/group` to selectively include and exclude entries from NIS. The primary usage of these entries is superseded by the name service switch, so the "+/-" form *may not be supported in future releases* .

If required, the "+/-" functionality can still be obtained for NIS by specifying `compat` as the source for `group` .

If the "+/-" functionality is required in conjunction with NIS+, specify both `compat` as the source for `group` and `nisplus` as the source for the pseudo-database `group_compat` . See `group(4)` , and `nsswitch.conf(4)` for details.

Solaris 2.4 and earlier releases provided definitions of the `getgrnam_r()` and `getgrgid_r()` functions as specified in POSIX.1c Draft 6. The final POSIX.1c standard changed the interface for these functions. Support for the Draft 6 interface is provided for compatibility only and may not be supported in future releases. New applications and libraries should use the POSIX standard interface.

For POSIX.1c-compliant applications, the `_POSIX_PTHREAD_SEMANTICS` and `_REENTRANT` flags are automatically turned on by defining the `_POSIX_C_SOURCE` flag with a value `>= 199506L`.

| | |
|--------------------|--|
| NAME | gethostbyname, gethostbyname_r, gethostbyaddr, gethostbyaddr_r, gethostent, gethostent_r, sethostent, endhostent – get network host entry |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] #include <netdb.h> struct hostent * gethostbyname(const char * <i>name</i>); struct hostent * gethostbyname_r(const char * <i>name</i>, struct hostent * <i>result</i>, char * <i>buffer</i>, int <i>buflen</i>, int * <i>h_errnop</i>); struct hostent * gethostbyaddr(const char * <i>addr</i>, int <i>len</i>, int <i>type</i>); struct hostent * gethostbyaddr_r(const char * <i>addr</i>, int <i>length</i>, int <i>type</i>, struct hostent * <i>result</i>, char * <i>buffer</i>, int <i>buflen</i>, int * <i>h_errnop</i>); struct hostent * gethostent(void); struct hostent * gethostent_r(struct hostent * <i>result</i>, char * <i>buffer</i>, int <i>buflen</i>, int * <i>h_errnop</i>); int sethostent(int <i>stayopen</i>); int endhostent(void); </pre> |
| DESCRIPTION | <p>These functions are used to obtain entries describing hosts. An entry may come from any of the sources for <code>hosts</code> specified in the <code>/etc/nsswitch.conf</code> file. See <code>nsswitch.conf(4)</code>.</p> <p>gethostbyname() searches for information for a host with the hostname specified by the character-string parameter <i>name</i>.</p> <p>gethostbyaddr() searches for information for a host with a given host address. The parameter <i>type</i> specifies the family of the address. This should be one of the address families defined in <code><sys/socket.h></code>. The parameter <i>addr</i> must be a pointer to a buffer containing the address. The address is given in a form specific to the address family. See the NOTES section below for more information. Also see the EXAMPLES section below on how to convert a “.” separated Internet IP address notation into the <i>addr</i> parameter. The parameter <i>len</i> specifies the length of the buffer indicated by <i>addr</i>.</p> |

All addresses are returned in network order. In order to interpret the addresses, `byteorder(3N)` must be used for byte order conversion.

The functions `sethostent()`, `gethostent()`, and `endhostent()` are used to enumerate host entries from the database.

`sethostent()` sets (or resets) the enumeration to the beginning of the set of host entries. This function should be called before the first call to `gethostent()`. Calls to `gethostbyname()` and `gethostbyaddr()` leave the enumeration position in an indeterminate state. If the `stayopen` flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to `endhostent()`.

Successive calls to `gethostent()` return either successive entries or `NULL`, indicating the end of the enumeration.

`endhostent()` may be called to indicate that the caller expects to do no further host entry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more host retrieval functions after calling `endhostent()`.

Reentrant Interfaces

The functions `gethostbyname()`, `gethostbyaddr()`, and `gethostent()` use static storage that is reused in each call, making these functions unsafe for use in multi-threaded applications.

The functions `gethostbyname_r()`, `gethostbyaddr_r()`, and `gethostent_r()` provide reentrant interfaces for these operations.

Each reentrant interface performs the same operation as its non-reentrant counterpart, named by removing the “`_r`” suffix. The reentrant interfaces, however, use buffers supplied by the caller to store returned results, and are safe for use in both single-threaded and multi-threaded applications.

Each reentrant interface takes the same parameters as its non-reentrant counterpart, as well as the following additional parameters. The parameter `result` must be a pointer to a `struct hostent` structure allocated by the caller. On successful completion, the function returns the host entry in this structure. The parameter `buffer` must be a pointer to a buffer supplied by the caller. This buffer is used as storage space for the host data. All of the pointers within the returned `struct hostent result` point to data stored within this buffer. See `RETURN VALUES`. The buffer must be large enough to hold all of the data associated with the host entry. The parameter `buflen` should give the size in bytes of the buffer indicated by `buffer`. The parameter `h_errnop` should be a pointer to an integer. An integer error status value is stored there on certain error conditions. See `ERRORS`.

For enumeration in multi-threaded applications, the position within the enumeration is a process-wide property shared by all threads. `sethostent()`

may be used in a multi-threaded application but resets the enumeration position for all threads. If multiple threads interleave calls to **gethostent_r()**, the threads will enumerate disjoint subsets of the host database.

Like their non-reentrant counterparts, **gethostbyname_r()** and **gethostbyaddr_r()** leave the enumeration position in an indeterminate state.

RETURN VALUES

Host entries are represented by the `struct hostent` structure defined in `<netdb.h>`:

```
struct hostent {
    char    *h_name;           /* canonical name of host */
    char    **h_aliases;      /* alias list */
    int     h_addrtype;       /* host address type */
    int     h_length;         /* length of address */
    char    **h_addr_list;    /* list of addresses */
};
```

See the `EXAMPLES` section below for information about how to retrieve a “.” separated Internet IP address string from the `h_addr_list` field of `struct hostent`.

The functions **gethostbyname()**, **gethostbyname_r()**, **gethostbyaddr()**, and **gethostbyaddr_r()** each return a pointer to a `struct hostent` if they successfully locate the requested entry; otherwise they return `NULL`.

The functions **gethostent()** and **gethostent_r()** each return a pointer to a `struct hostent` if they successfully enumerate an entry; otherwise they return `NULL`, indicating the end of the enumeration.

The functions **gethostbyname()**, **gethostbyaddr()**, and **gethostent()** use static storage, so returned data must be copied before a subsequent call to any of these functions if the data is to be saved.

When the pointer returned by the reentrant functions **gethostbyname_r()**, **gethostbyaddr_r()**, and **gethostent_r()** is not `NULL`, it is always equal to the `result` pointer that was supplied by the caller.

The functions **sethostent()** and **endhostent()** return 0 on success.

ERRORS

The reentrant functions **gethostbyname_r()**, **gethostbyaddr_r()**, and **gethostent_r()** will return `NULL` and set `errno` to `ERANGE` if the length of the buffer supplied by caller is not large enough to store the result. See `Intro(2)` for the proper usage and interpretation of `errno` in multithreaded applications.

The reentrant functions **gethostbyname_r()** and **gethostbyaddr_r()** set the integer pointed to by `h_errnop` to one of these values in case of error.

On failures, the non-reentrant functions **gethostbyname()** and **gethostbyaddr()** set a global integer *h_errno* to indicate one of these error codes (defined in `<netdb.h>`): `HOST_NOT_FOUND`, `TRY_AGAIN`, `NO_RECOVERY`, `NO_DATA`, and `NO_ADDRESS`.

Note however that if a resolver is provided with a malformed address, or if any other error occurs before **gethostbyname()** is resolved, then **gethostbyname()** returns an internal error with a value of `-1`.

gethostbyname() will set *h_errno* to `NETDB_INTERNAL` when it returns a `NULL` value.

EXAMPLES

EXAMPLE 1 Using **gethostbyname()**

Here is a sample program that gets the canonical name, aliases, and “.” separated Internet IP addresses for a given “.” separated IP address:

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
main(int argc, const char **argv)
{
    \011ulong_t addr;
    \011struct hostent *hp;
    \011char **p;
    \011if (argc != 2) {
    \011    (void) printf("usage: %s IP-address\n",
    \011    argv[0]);
    \011    exit (1);
    \011}
    \011if ((int)(addr = inet_addr(argv[1])) == -1) {
    \011    (void) printf("IP-address must be of the form a.b.c.d\n",
    \011    );
    \011    exit (2);
    \011}
    \011hp = gethostbyaddr((char *)&addr, sizeof (addr), AF_INET);
    \011if (hp == NULL) {
    \011    (void) printf("host information for %s not found\n",
    \011    argv[1]);
    \011    exit (3);
    \011}
    \011for (p = hp->h_addr_list; *p != 0; p++) {
    \011    struct in_addr in;
    \011    char **q;
    \011    (void) memcpy(&in.s_addr, *p, sizeof (in.s_addr));
    \011    (void) printf("%s\t%s", inet_ntoa(in), hp->h_name);
    \011    for (q = hp->h_aliases; *q != 0; q++)
    \011        (void) printf(" %s", *q);
    \011    (void) putchar('\n');
    \011}
}
```

```
\011exit (0);
}
```

Note that the above sample program is unsafe for use in multithreaded applications.

FILES

```
/etc/hosts
/etc/netconfig
/etc/nsswitch.conf
```

ATTRIBUTES

See **attributes** (5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|---|
| MT-Level | See "Reentrant Interfaces" in DESCRIPTION . |

SEE ALSO

Intro(2), **Intro**(3), **byteorder**(3N), **inet**(3N), **netdir**(3N), **hosts**(4), **netconfig**(4), **nsswitch.conf**(4), **attributes**(5), **netdb**(5)

WARNINGS

The reentrant interfaces **gethostbyname_r()**, **gethostbyaddr_r()**, and **gethostent_r()** are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.

NOTES

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

In order to ensure that they all return consistent results, **gethostbyname()**, **gethostbyname_r()**, and **netdir_getbyname()** are implemented in terms of the same internal library function. This function obtains the system-wide source lookup policy based on the **inet** family entries in **netconfig**(4) and the **hosts:** entry in **nsswitch.conf**(4). Similarly, **gethostbyaddr()**, **gethostbyaddr_r()**, and **netdir_getbyaddr()** are implemented in terms of the same internal library function. If the **inet** family entries in **netconfig**(4) have a "-" in the last column for nametoaddr libraries, then the entry for **hosts** in **nsswitch.conf** will be used; otherwise the nametoaddr libraries in that column will be used, and **nsswitch.conf** will not be consulted.

There is no analogue of **gethostent()** and **gethostent_r()** in the **netdir** functions, so these enumeration functions go straight to the **hosts** entry in **nsswitch.conf**. Thus enumeration may return results from a different

source than that used by `gethostbyname()` , `gethostbyname_r()` , `gethostbyaddr()` , and `gethostbyaddr_r()` .

All the functions that return a `struct hostent` must always return the *canonical name* in the `h_name` field. This name, by definition, is the well-known and official hostname shared between all aliases and all addresses. The underlying source that satisfies the request determines the mapping of the input name or address into the set of names and addresses in `hostent` . Different sources might do that in different ways. If there is more than one alias and more than one address in `hostent` , no pairing is implied between them.

The system will strive to put the addresses on the same subnet as that of the caller first.

When compiling multi-threaded applications, see `Intro(3)` , *Notes On Multithread Applications* , for information about the use of the `_REENTRANT` flag.

Use of the enumeration interfaces `gethostent()` and `gethostent_r()` is discouraged; enumeration may not be supported for all database sources. The semantics of enumeration are discussed further in `nsswitch.conf(4)` .

The current implementations of these functions only return or accept addresses for the Internet address family (type `AF_INET`) .

The form for an address of type `AF_INET` is a `struct in_addr` defined in `<netinet/in.h>` . The functions described in `inet(3N)` , and illustrated in the `EXAMPLES` section above, are helpful in constructing and manipulating addresses in this form.

| | |
|--------------------|---|
| NAME | gethostid – get unique identifier of current host |
| SYNOPSIS | <pre>#include <unistd.h> long gethostid(void);</pre> |
| DESCRIPTION | The gethostid() function returns the 32-bit identifier for the current host, which should be unique across all hosts. This number is usually taken from the CPU board's ID PROM. |
| SEE ALSO | hostid(1) , sysinfo(2) |

| | |
|----------------------|--|
| NAME | gethostname, sethostname – get or set name of current host |
| SYNOPSIS | <pre>#include <unistd.h> int gethostname(char * name, int namelen); int sethostname(char * name, int namelen);</pre> |
| DESCRIPTION | <p>The gethostname() function returns the standard host name for the current processor, as previously set by sethostname() . The <i>namelen</i> argument specifies the size of the array pointed to by <i>name</i> . The returned name is null-terminated unless insufficient space is provided.</p> <p>The sethostname() function sets the name of the host machine to be <i>name</i> , which has length <i>namelen</i> . This call is restricted to the super-user and is normally used only when the system is bootstrapped.</p> <p>Host names are limited to MAXHOSTNAMELEN characters, currently 256, defined in the <netdb.h> header.</p> |
| RETURN VALUES | Upon successful completion, gethostname() and sethostname() return 0 . Otherwise, they return -1 and set <i>errno</i> to indicate the error. |
| ERRORS | The gethostname() and sethostname() functions will fail if: EFAULT The <i>name</i> or <i>namelen</i> argument gave an invalid address. The sethostname() function will fail if: EPERM The caller was not the super-user. |
| SEE ALSO | sysinfo(2) , uname(2) , gethostid(3C) |

NAME | gethostname – get name of current host

SYNOPSIS |

```
cc [ flag ... ] file ... -lxnet [ library ... ]
#include <unistd.h>

int gethostname(char *name, size_t namelen);
```

DESCRIPTION | The **gethostname()** function returns the standard host name for the current machine. The *namelen* argument specifies the size of the array pointed to by the *name* argument. The returned name is null-terminated, except that if *namelen* is an insufficient length to hold the host name, then the returned name is truncated and it is unspecified whether the returned name is null-terminated.

Host names are limited to 255 bytes.

RETURN VALUES | On successful completion, 0 is returned. Otherwise, -1 is returned.

ERRORS | No errors are defined.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **uname(1)**, **gethostid(3C)**, **attributes(5)**

| NAME | gethrtime, gethrvtime – get high resolution time | | | | |
|--------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <sys/time.h> hrtime_t gethrtime(void); hrtime_t gethrvtime(void);</pre> | | | | |
| DESCRIPTION | <p>The gethrtime() function returns the current high-resolution real time. Time is expressed as nanoseconds since some arbitrary time in the past; it is not correlated in any way to the time of day, and thus is <i>not</i> subject to resetting or drifting by way of adjtime(2) or settimeofday(3C). The hi-res timer is ideally suited to performance measurement tasks, where cheap, accurate interval timing is required.</p> <p>The gethrvtime() function returns the current high-resolution LWP virtual time, expressed as total nanoseconds of execution time. This function requires that micro state accounting be enabled with the <code>p_time</code> utility (see proc(1)).</p> <p>The gethrtime() and gethrvtime() functions both return an <code>hrtime_t</code>, which is a 64-bit (<code>long long</code>)signed integer.</p> | | | | |
| EXAMPLES | <p>The following code fragment measures the average cost of getpid(2) :</p> <pre>hrtime_t start, end; int i, iters = 100; start = gethrtime(); for (i = 0; i < iters; i++) getpid(); end = gethrtime(); printf("Avg getpid() time = %lld nsec\ ", (end - start) / iters);</pre> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | proc(1) , adjtime(2) , gettimeofday(3C) , settimeofday(3C) , attributes(5) | | | | |
| NOTES | Although the units of hi-res time are always the same (nanoseconds), the actual resolution is hardware dependent. Hi-res time is guaranteed to be | | | | |

monotonic (it won't go backward, it won't periodically wrap) and linear (it won't occasionally speed up or slow down for adjustment, like the time of day can), but not necessarily unique: two sufficiently proximate calls may return the same value.

NAME getloadavg – get system load averages

SYNOPSIS #include <sys/loadavg.h>

```
int getloadavg(double loadavg[], int nelem);
```

DESCRIPTION The **getloadavg()** function returns the number of processes in the system run queue averaged over various periods of time. Up to *nelem* samples are retrieved and assigned to successive elements of *loadavg*[]. The system imposes a maximum of 3 samples, representing averages over the last 1, 5, and 15 minutes, respectively. The `LOADAVG_1MIN`, `LOADAVG_5MIN`, and `LOADAVG_15MIN` indices, defined in `<sys/loadavg.h>`, can be used to extract the data from the appropriate element of the *loadavg*[] array.

RETURN VALUES Upon successful completion, the number of samples actually retrieved is returned. If the load average was unobtainable, -1 is returned and `errno` is set to indicate the error.

ERRORS The **getloadavg()** function will fail if:

EINVAL The number of elements specified is less than zero.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **uptime(1)**, **w(1)**, **kstat(3K)**, **standards(5)**

| | |
|----------------------|---|
| NAME | getlogin, getlogin_r – get login name |
| SYNOPSIS | <pre>#include <unistd.h> char * getlogin(void); char * getlogin_r(char * name, int namelen);</pre> |
| POSIX | <pre>cc [flag ...] file ... -D_POSIX_PTHREAD_SEMANTICS [library ...] int getlogin_r(char * name, size_t namesize);</pre> |
| DESCRIPTION | <p>The getlogin() function returns a pointer to the login name as found in <code>/var/adm/utmp</code>. It may be used in conjunction with getpwnam(3C) to locate the correct password file entry when the same user ID is shared by several login names.</p> <p>If getlogin() is called within a process that is not attached to a terminal, it returns a null pointer. The correct procedure for determining the login name is to call cuserid(3S), or to call getlogin() and if it fails to call getpwuid(3C).</p> <p>The getlogin_r() function has the same functionality as getlogin() except that the caller must supply a buffer <i>name</i> with length <i>namelen</i> to store the result. The <i>name</i> buffer must be at least <code>_POSIX_LOGIN_NAME_MAX</code> bytes in size (defined in <code><limits.h></code>). The POSIX version (see standards(5)) of getlogin_r() takes a <i>namesize</i> parameter of type <code>size_t</code>.</p> |
| RETURN VALUES | <p>Upon successful completion, getlogin() returns a pointer to the login name or a null pointer if the user's login name cannot be found. Otherwise it returns a null pointer and sets <code>errno</code> to indicate the error.</p> <p>The POSIX getlogin_r() returns 0 if successful, or the error number upon failure.</p> |
| ERRORS | <p>The getlogin() function may fail if:</p> <p>EMFILE There are <code>OPEN_MAX</code> file descriptors currently open in the calling process.</p> <p>ENFILE The maximum allowable number of files is currently open in the system.</p> |

ENXIO The calling process has no controlling terminal.
The **getlogin_r()** function will fail if:

ERANGE The size of the buffer is smaller than the result to be returned.

EINVAL And entry for the current user was not found in the `/var/adm/utmp` file.

USAGE The return value may point to static data whose content is overwritten by each call.

Three names associated with the current process can be determined: `getpwuid(geteuid())` returns the name associated with the effective user ID of the process; **getlogin()** returns the name associated with the current login activity; and `getpwuid(getuid())` returns the name associated with the real user ID of the process.

FILES

`/var/adm/utmp` accounting file

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | See NOTES below. |

SEE ALSO

geteuid(2), **getuid(2)**, **cuserid(3S)**, **getgrnam(3C)**, **getpwnam(3C)**, **getpwuid(3C)**, **utmp(4)**, **attributes(5)**, **standards(5)**

NOTES

When compiling multithreaded programs, see **Intro(3)**, *Notes On Multithreaded Applications*.

The **getlogin()** function is unsafe in multithreaded applications. The **getlogin_r()** function should be used instead.

Solaris 2.4 and earlier releases provided a **getlogin_r()** as specified in POSIX.1c Draft 6. The final POSIX.1c standard changed the interface as described above. Support for the Draft 6 interface is provided for compatibility only and may not be supported in future releases. New applications and libraries should use the POSIX standard interface.

| | |
|--|--|
| NAME | getmntent, getmntany, hasmntopt, putmntent – get mnttab file information |
| SYNOPSIS | <pre>#include <stdio.h> #include <sys/mnttab.h> int getmntent(FILE * fp, struct mnttab * mp); int getmntany(FILE * fp, struct mnttab * mp, struct mnttab * mpref); char * hasmntopt(struct mnttab * mnt, char * opt); int putmntent(FILE * iop, struct mnttab * mp);</pre> |
| DESCRIPTION | |
| getmntent() and getmntany() | <p>The getmntent() and getmntany() functions each fill in the structure pointed to by <i>mp</i> with the broken-out fields of a line in the <code>/etc/mnttab</code> file. Each line in the file contains a <code>mnttab</code> structure, which is defined in the <code><sys/mnttab.h></code> header. The structure contains the following members which are described on the <code>mnttab(4)</code> manual page.</p> <pre>char *mnt_special; char *mnt_mountp; char *mnt_fstype; char *mnt_mntopts; char *mnt_time;</pre> <p>The getmntent() function returns a pointer to the next <code>mnttab</code> structure in the file. Successive calls can be used to search the entire file. The getmntany() function searches the file referenced by <i>fp</i> until a match is found between a line in the file and <i>mpref</i>. A match occurs if all non-null entries in <i>mpref</i> match the corresponding fields in the file. Note that these routines do not open, close, or rewind the file.</p> <p>Applications requiring consistent access to <code>/etc/mnttab</code> should lock the file. The following C code segment applies a shared file segment lock to the file. Since it a shared lock, multiple processes are allowed to read the file at the same time, but no process will be allowed to modify its contents until all of the shared locks have been released and an exclusive lock has been acquired.</p> <pre>#include <fcntl.h> struct flock lb; lb.l_type = F_RDLCK;</pre> |

```

lb.l_whence = 0;
lb.l_start = 0;
lb.l_len = 0

fcntl(fileno(fp), F_SETLK, &lb);

```

The data pointed to by the `mnttab` structure members are stored in a static area and must be copied to be saved between successive calls.

hasmntopt()

The **hasmntopt()** function scans the `mnt_mntopts` member of the `mnttab` structure `mnt` for a substring that matches `opt`. It returns the address of the substring if a match is found; otherwise it returns 0.

putmntent()

The **putmntent()** macro formats the contents of the `mnttab` structure according to the layout required for the `/etc/mnttab` file and writes the entry to the file. The file should be opened in append mode (`fopen(3S)` with an "a" mode) so that the entry is appended to the file.

Applications writing to `/etc/mnttab` should lock the file to provide consistent access for other applications. The following C code segment applies an exclusive file segment lock to the file. This exclusive lock guarantees that no other processes are updating or reading from `/etc/mnttab`.

```

lb.l_type = F_WRLCK;
lb.l_whence = 0;

lb.l_start = 0;
lb.l_len = 0;

fcntl(fileno(fp), F_SETLK, &lb);

```

RETURN VALUES**getmntent() and
getmntany()**

If the next entry is successfully read by **getmntent()** or a match is found with **getmntany()**, 0 is returned. If an EOF is encountered on reading, these functions return -1. If an error is encountered, a value greater than 0 is returned. The following error values are defined in `<sys/mnttab.h>`:

MNT_TOOLONG A line in the file exceeded the internal buffer size of `MNT_LINE_MAX`.

MNT_TOOMANY A line in the file contains too many fields.

MNT_TOOFEW A line in the file contains too few fields.

hasmntopt() Upon successful completion, **hasmntopt()** returns the address of the substring if a match is found. Otherwise, it returns 0 .

putmntent() Upon successful completion, **putmntent()** returns the number of bytes printed to the specified file. Otherwise, it returns EOF .

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO **mnttab(4)** , **attributes(5)**

| | |
|--------------------|---|
| NAME | getnetbyname, getnetbyname_r, getnetbyaddr, getnetbyaddr_r, getnetent, getnetent_r, setnetent, endnetent – get network entry |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lsocket -lnsl [<i>library</i> ...] #include <netdb.h> struct netent * getnetbyname(const char * <i>name</i>); struct netent * getnetbyname_r(const char * <i>name</i>, struct netent * <i>result</i>, char * <i>buffer</i>, int <i>buflen</i>); struct netent * getnetbyaddr(long <i>net</i>, int <i>type</i>); struct netent * getnetbyaddr_r(long <i>net</i>, int <i>type</i>, struct netent * <i>result</i>, char * <i>buffer</i>, int <i>buflen</i>); struct netent * getnetent(void); struct netent * getnetent_r(struct netent * <i>result</i>, char * <i>buffer</i>, int <i>buflen</i>); int setnetent(int <i>stayopen</i>); int endnetent(void);</pre> |
| DESCRIPTION | <p>These functions are used to obtain entries for networks. An entry may come from any of the sources for networks specified in the <code>/etc/nsswitch.conf</code> file. See <code>nsswitch.conf(4)</code>.</p> <p>getnetbyname() searches for a network entry with the network name specified by the character string parameter <i>name</i>.</p> <p>getnetbyaddr() searches for a network entry with the network address specified by <i>net</i>. The parameter <i>type</i> specifies the family of the address. This should be one of the address families defined in <code><sys/socket.h></code>. See the NOTES section below for more information.</p> <p>All addresses are returned in network order. In order to interpret the addresses, <code>byteorder(3N)</code> must be used for byte order conversion.</p> |

The functions **setnetent()** , **getnetent()** , and **endnetent()** are used to enumerate network entries from the database.

setnetent() sets (or resets) the enumeration to the beginning of the set of network entries. This function should be called before the first call to **getnetent()** . Calls to **getnetbyname()** and **getnetbyaddr()** leave the enumeration position in an indeterminate state. If the *stayopen* flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to **endnetent()** .

Successive calls to **getnetent()** return either successive entries or NULL, indicating the end of the enumeration.

endnetent() may be called to indicate that the caller expects to do no further network entry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more network entry retrieval functions after calling **endnetent()** .

Reentrant Interfaces

The functions **getnetbyname()** , **getnetbyaddr()** , and **getnetent()** use static storage that is reused in each call, making these routines unsafe for use in multi-threaded applications.

The functions **getnetbyname_r()** , **getnetbyaddr_r()** , and **getnetent_r()** provide reentrant interfaces for these operations.

Each reentrant interface performs the same operation as its non-reentrant counterpart, named by removing the “_r” suffix. The reentrant interfaces, however, use buffers supplied by the caller to store returned results, and are safe for use in both single-threaded and multi-threaded applications.

Each reentrant interface takes the same parameters as its non-reentrant counterpart, as well as the following additional parameters. The parameter *result* must be a pointer to a `struct netent` structure allocated by the caller. On successful completion, the function returns the network entry in this structure. The parameter *buffer* must be a pointer to a buffer supplied by the caller. This buffer is used as storage space for the network entry data. All of the pointers within the returned `struct netent` *result* point to data stored within this buffer. See RETURN VALUES . The buffer must be large enough to hold all of the data associated with the network entry. The parameter *buflen* should give the size in bytes of the buffer indicated by *buffer* .

For enumeration in multi-threaded applications, the position within the enumeration is a process-wide property shared by all threads. **setnetent()** may be used in a multi-threaded application but resets the enumeration position for all threads. If multiple threads interleave calls to **getnetent_r()** , the threads will enumerate disjointed subsets of the network database.

RETURN VALUES

Like their non-reentrant counterparts, **getnetbyname_r()** and **getnetbyaddr_r()** leave the enumeration position in an indeterminate state.

Network entries are represented by the `struct netent` structure defined in `<netdb.h>`.

The functions **getnetbyname()**, **getnetbyname_r()**, **getnetbyaddr()**, and **getnetbyaddr_r()** each return a pointer to a `struct netent` if they successfully locate the requested entry; otherwise they return `NULL`.

The functions **getnetent()** and **getnetent_r()** each return a pointer to a `struct netent` if they successfully enumerate an entry; otherwise they return `NULL`, indicating the end of the enumeration.

The functions **getnetbyname()**, **getnetbyaddr()**, and **getnetent()** use static storage, so returned data must be copied before a subsequent call to any of these functions if the data is to be saved.

When the pointer returned by the reentrant functions **getnetbyname_r()**, **getnetbyaddr_r()**, and **getnetent_r()** is non-`NULL`, it is always equal to the *result* pointer that was supplied by the caller.

The functions **setnetent()** and **endnetent()** return 0 on success.

ERRORS

The reentrant functions **getnetbyname_r()**, **getnetbyaddr_r()** and **getnetent_r()** will return `NULL` and set *errno* to `ERANGE` if the length of the buffer supplied by caller is not large enough to store the result. See **intro(2)** for the proper usage and interpretation of *errno* in multi-threaded applications.

FILES

`/etc/networks`
`/etc/nsswitch.conf`

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

Intro(2), **Intro(3)**, **byteorder(3N)**, **inet(3N)**, **networks(4)**, **nsswitch.conf(4)**, **attributes(5)**, **netdb(5)**

WARNINGS

The reentrant interfaces **getnetbyname_r()**, **getnetbyaddr_r()**, and **getnetent_r()** are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.

NOTES

The current implementation of these functions only return or accept network numbers for the Internet address family (type `AF_INET`). The functions described in `inet(3N)` may be helpful in constructing and manipulating addresses and network numbers in this form.

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multi-threaded applications, see `Intro(3)`, *Notes On Multithread Applications*, for information about the use of the `_REENTRANT` flag.

Use of the enumeration interfaces `getnetent()` and `getnetent_r()` is discouraged; enumeration may not be supported for all database sources. The semantics of enumeration are discussed further in `nsswitch.conf(4)`.

| | |
|--------------------|---|
| NAME | getnetconfig, setnetconfig, endnetconfig, getnetconfigent, freenetconfigent, nc_perror, nc_spperror – get network configuration database entry |
| SYNOPSIS | <pre>#include <netconfig.h> struct netconfig * getnetconfig(void * handlep); void * setnetconfig(void); int endnetconfig(void * handlep); struct netconfig * getnetconfigent(const char * netid); void freenetconfigent(struct netconfig * netconfigp); void nc_perror(const char * msg); char * nc_spperror(void);</pre> |
| DESCRIPTION | <p>The library routines described on this page are part of the Network Selection component. They provide the application access to the system network configuration database, <code>/etc/netconfig</code>. In addition to the routines for accessing the <code>netconfig</code> database, Network Selection includes the environment variable <code>NETPATH</code> (see <code>environ(5)</code>) and the <code>NETPATH</code> access routines described in <code>getnetpath(3N)</code>.</p> <p>getnetconfig() returns a pointer to the current entry in the <code>netconfig</code> database, formatted as a <code>struct netconfig</code>. Successive calls will return successive <code>netconfig</code> entries in the <code>netconfig</code> database. getnetconfig() can be used to search the entire <code>netconfig</code> file. getnetconfig() returns NULL at the end of the file. <i>handlep</i> is the handle obtained through setnetconfig().</p> <p>A call to setnetconfig() has the effect of “binding” to or “rewinding” the <code>netconfig</code> database. setnetconfig() must be called before the first call to getnetconfig() and may be called at any other time. setnetconfig() need <i>not</i> be called before a call to getnetconfigent(). setnetconfig() returns a unique handle to be used by getnetconfig().</p> <p>endnetconfig() should be called when processing is complete to release resources for reuse. <i>handlep</i> is the handle obtained through setnetconfig(). Programmers should be aware, however, that the last call to endnetconfig() frees all memory allocated by getnetconfig() for the <code>struct netconfig</code> data structure. endnetconfig() may not be called before setnetconfig().</p> <p>getnetconfigent() returns a pointer to the <code>struct netconfig</code> structure corresponding to <i>netid</i>. It returns NULL if <i>netid</i> is invalid (that is, does not name an entry in the <code>netconfig</code> database).</p> |

freenetconfig() frees the netconfig structure pointed to by *netconfig* (previously returned by **getnetconfig()**).

nc_perror() prints a message to the standard error indicating why any of the above routines failed. The message is prepended with the string *msg* and a colon. A NEWLINE is appended at the end of the message.

nc_spperror() is similar to **nc_perror()** but instead of sending the message to the standard error, will return a pointer to a string that contains the error message.

nc_perror() and **nc_spperror()** can also be used with the NETPATH access routines defined in **getnetpath(3N)** .

RETURN VALUES

setnetconfig() returns a unique handle to be used by **getnetconfig()** . In the case of an error, **setnetconfig()** returns NULL and **nc_perror()** or **nc_spperror()** can be used to print the reason for failure.

getnetconfig() returns a pointer to the current entry in the **netconfig()** database, formatted as a `struct netconfig` . **getnetconfig()** returns NULL at the end of the file, or upon failure.

endnetconfig() returns 0 on success and -1 on failure (for example, if **setnetconfig()** was not called previously).

On success, **getnetconfig()** returns a pointer to the `struct netconfig` structure corresponding to *netid* ; otherwise it returns NULL.

nc_spperror() returns a pointer to a buffer which contains the error message string. This buffer is overwritten on each call. In multithreaded applications, this buffer is implemented as thread-specific data.

ATTRIBUTES

See **attributes** (5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

getnetpath(3N) , **netconfig(4)** , **attributes(5)** , **environ(5)**

ONC+ Developer's Guide Transport Interfaces Programming Guide

| | |
|--------------------|---|
| NAME | getnetgrent, getnetgrent_r, setnetgrent, endnetgrent, innetgr – get network group entry |
| SYNOPSIS | <pre>#include <netdb.h> int getnetgrent(char ** machinep, char ** userp, char ** domainp); int getnetgrent_r(char ** machinep, char ** userp, char ** domainp, char * buffer, int buflen); int setnetgrent(const char * netgroup); int endnetgrent(void); int innetgr(const char * netgroup, const char * machine, const char * user, const char * domain);</pre> |
| DESCRIPTION | <p>These functions are used to test membership in and enumerate members of “netgroup” network groups defined in a system database. Netgroups are sets of (machine,user,domain) triples (see netgroup(4)).</p> <p>These functions consult the source specified for <code>netgroup</code> in the <code>/etc/nsswitch.conf</code> file (see nsswitch.conf(4)).</p> <p>The function innetgr() returns 1 if there is a netgroup <code>netgroup</code> that contains the specified <code>machine</code>, <code>user</code>, <code>domain</code> triple as a member; otherwise it returns 0 . Any of the supplied pointers <code>machine</code> , <code>user</code> , and <code>domain</code> may be <code>NULL</code> , signifying a "wild card" that matches all values in that position of the triple.</p> <p>The innetgr() function is safe for use in single-threaded and multithreaded applications.</p> <p>The functions setnetgrent() , getnetgrent() , and endnetgrent() are used to enumerate the members of a given network group.</p> <p>The function setnetgrent() establishes the network group specified in the parameter <code>netgroup</code> as the current group whose members are to be enumerated.</p> <p>Successive calls to the function getnetgrent() will enumerate the members of the group established by calling setnetgrent() ; each call returns 1 if it succeeds in obtaining another member of the network group, or 0 if there are no further members of the group.</p> <p>When calling either getnetgrent() or getnetgrent_r() , addresses of the three character pointers are used as arguments, for example:</p> <pre>char *mp</pre> |

```

'
*up
'
*dp
;
getnetgrent(
&mp
'
&up
'
&dp
);

```

Upon successful return from **getnetgrent()**, the pointer *mp* points to a string containing the name of the machine part of the member triple, *up* points to a string containing the user name and *dp* points to a string containing the domain name. If the pointer returned for *mp*, *up*, or *dp* is `NULL`, it signifies that the element of the netgroup contains wild card specifier in that position of the triple.

The pointers returned by **getnetgrent()** point into a buffer allocated by **setnetgrent()** that is reused by each call. This space is released when an **endnetgrent()** call is made, and should not be released by the caller. This implementation is not safe for use in multi-threaded applications.

The function **getnetgrent_r()** is similar to **getnetgrent()** function, but it uses a buffer supplied by the caller for the space needed to store the results. The parameter *buffer* should be a pointer to a buffer allocated by the caller and the length of this buffer should be specified by the parameter *buflen*. The buffer must be large enough to hold the data associated with the triple. The **getnetgrent_r()** function is safe for use both in single-threaded and multi-threaded applications.

The function **endnetgrent()** frees the space allocated by the previous **setnetgrent()** call. The equivalent of an **endnetgrent()** implicitly performed whenever a **setnetgrent()** call is made to a new network group.

Note that while **setnetgrent()** and **endnetgrent()** are safe for use in multi-threaded applications, the effect of each is process-wide. Calling **setnetgrent()** resets the enumeration position for all threads. If multiple threads interleave calls to **getnetgrent_r()** each will enumerate a disjoint subset of the netgroup. Thus the effective use of these functions in multi-threaded applications may require coordination by the caller.

ERRORS

The function **getnetgrent_r()** will return 0 and set `errno` to `ERANGE` if the length of the buffer supplied by caller is not large enough to store the result.

See **Intro(2)** for the proper usage and interpretation of `errno` in multi-threaded applications.

The functions **setnetgrent()** and **endnetgrent()** return 0 upon success.

FILES

`/etc/nsswitch.conf`

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|---------------------------------|
| MT-Level | See DESCRIPTION section. |

SEE ALSO

Intro(2) , **Intro(3)** , **netgroup(4)** , **nsswitch.conf(4)** , **attributes(5)**

WARNINGS

The function **getnetgrent_r()** is included in this release on an uncommitted basis only, and is subject to change or removal in future minor releases.

NOTES

Only the Network Information Services, NIS and NIS+, are supported as sources for the `netgroup` database.

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multi-threaded applications, see **Intro(3)** , *Notes On Multithread Applications* , for information about the use of the `_REENTRANT` flag.

| | |
|----------------------|--|
| NAME | getnetpath, setnetpath, endnetpath – get /etc/netconfig entry corresponding to NETPATH component |
| SYNOPSIS | <pre>#include <netconfig.h> struct netconfig * getnetpath(void * handlep); void * setnetpath(void); int endnetpath(void * handlep);</pre> |
| DESCRIPTION | <p>The routines described on this page are part of the Network Selection component. They provide the application access to the system network configuration database, /etc/netconfig, as it is "filtered" by the NETPATH environment variable. See environ(5). See getnetconfig(3N) for other routines that also access the network configuration database directly. The NETPATH variable is a list of colon-separated network identifiers.</p> <p>getnetpath() returns a pointer to the netconfig database entry corresponding to the first valid NETPATH component. The netconfig entry is formatted as a struct netconfig. On each subsequent call, getnetpath() returns a pointer to the netconfig entry that corresponds to the next valid NETPATH component. getnetpath() can thus be used to search the netconfig database for all networks included in the NETPATH variable. When NETPATH has been exhausted, getnetpath() returns NULL.</p> <p>A call to setnetpath() "binds" to or "rewinds" NETPATH. setnetpath() must be called before the first call to getnetpath() and may be called at any other time. It returns a handle that is used by getnetpath().</p> <p>getnetpath() silently ignores invalid NETPATH components. A NETPATH component is invalid if there is no corresponding entry in the netconfig database.</p> <p>If the NETPATH variable is unset, getnetpath() behaves as if NETPATH were set to the sequence of "default" or "visible" networks in the netconfig database, in the order in which they are listed.</p> <p>endnetpath() may be called to "unbind" from NETPATH when processing is complete, releasing resources for reuse. Programmers should be aware, however, that endnetpath() frees all memory allocated by getnetpath() for the struct netconfig data structure. endnetpath() returns 0 on success and -1 on failure (for example, if setnetpath() was not called previously).</p> |
| RETURN VALUES | <p>setnetpath() returns a handle that is used by getnetpath(). In case of an error, setnetpath() returns NULL. nc_perror() or nc_spperror() can be used to print out the reason for failure. See getnetconfig(3N).</p> |

When first called, **getnetpath()** returns a pointer to the `netconfig` database entry corresponding to the first valid NETPATH component. When NETPATH has been exhausted, **getnetpath()** returns NULL.

endnetpath() returns 0 on success and -1 on failure (for example, if **setnetpath()** was not called previously).

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

getnetconfig(3N), **netconfig(4)**, **attributes(5)**, **environ(5)**

ONC+ Developer's Guide Transport Interfaces Programming Guide

| | | | | | | | | | | | |
|--------------------|---|-------------------|--|-----------------|---|-----------------|--|-----------------|---|-------------------|---|
| NAME | getnstr, getstr, mvgetnstr, mvgetstr, mvwgetnstr, mvwgetstr, wgetnstr, wgetstr – get a multibyte character string from terminal | | | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int getnstr(char * str, int n); int getstr(char * str); int mvgetnstr(int y, int x, char * str, int n); int mvgetstr(int y, int x, char * str); int mvwgetnstr(WINDOW * win, int y, int x, char * str, int n); int mvwgetstr(WINDOW * win, int y, int x, char * str); int wgetnstr(WINDOW * win, char * str, int n); int wgetstr(WINDOW * win, char * str);</pre> | | | | | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="vertical-align: top;"><i>str</i></td> <td>Is a pointer to the area where the character string is to be placed.</td> </tr> <tr> <td style="vertical-align: top;"><i>n</i></td> <td>Is the maximum number of characters to read from input.</td> </tr> <tr> <td style="vertical-align: top;"><i>y</i></td> <td>Is the y (row) coordinate of starting position of character string to be read.</td> </tr> <tr> <td style="vertical-align: top;"><i>x</i></td> <td>Is the x (column) coordinate of starting position of character string to be read.</td> </tr> <tr> <td style="vertical-align: top;"><i>win</i></td> <td>Points to the window associated with the terminal from which the character is to be read.</td> </tr> </table> | <i>str</i> | Is a pointer to the area where the character string is to be placed. | <i>n</i> | Is the maximum number of characters to read from input. | <i>y</i> | Is the y (row) coordinate of starting position of character string to be read. | <i>x</i> | Is the x (column) coordinate of starting position of character string to be read. | <i>win</i> | Points to the window associated with the terminal from which the character is to be read. |
| <i>str</i> | Is a pointer to the area where the character string is to be placed. | | | | | | | | | | |
| <i>n</i> | Is the maximum number of characters to read from input. | | | | | | | | | | |
| <i>y</i> | Is the y (row) coordinate of starting position of character string to be read. | | | | | | | | | | |
| <i>x</i> | Is the x (column) coordinate of starting position of character string to be read. | | | | | | | | | | |
| <i>win</i> | Points to the window associated with the terminal from which the character is to be read. | | | | | | | | | | |
| DESCRIPTION | <p>The getstr() and wgetstr() functions get a character string from the terminal associated with the window <code>stdscr</code> or window <code>win</code>, respectively. The mvgetstr() and mvwgetstr() functions move the cursor to the position specified in <code>stdscr</code> or <code>win</code>, respectively, then get a character string.</p> <p>These functions call wgetch(3XC) and place each received character in <code>str</code> until a newline is received, which is also placed in <code>str</code>. The erase and kill characters set by the user are processed.</p> <p>The getnstr(), mvgetnstr(), mvwgetnstr() and wgetnstr() functions read at most <code>n</code> characters. These functions are used to prevent overflowing the input buffer.</p> | | | | | | | | | | |

The `getnstr()` , `wgetnstr()` , `mvgetnstr()` , and `mvwgetnstr()` functions only return complete multibyte characters. If the area pointed to by `str` is not large enough to hold at least one character, these functions fail.

RETURN VALUES

On success, these functions return `OK` . Otherwise, they return `ERR` .

ERRORS

None.

SEE ALSO

`getch(3XC)`

| | |
|--------------------|--|
| NAME | getn_wstr, get_wstr, mvgetn_wstr, mvget_wstr, mvwgetn_wstr, mvwget_wstr, wgetn_wstr, wget_wstr – get a wide character string from terminal |
| SYNOPSIS | <pre>#include <curses.h> int getn_wstr(wint_t * wstr, int , int get_wstr(wint_t * wstr); int mvgetn_wstr(int y, int x, wint_t * wstr, int n); int mvget_wstr(int y, int x, wint_t * wstr); int mvwgetn_wstr(WINDOW * win, int y, int x, wint_t * wstr, int n); int mvwget_wstr(WINDOW * win, int y, int x, wint_t * wstr); int wgetn_wstr(WINDOW * win, wint_t * wstr, int n); int wget_wstr(WINDOW * win, wint_t * wstr);</pre> |
| PARAMETERS | <p>wstr Is a pointer to the area where the character string is to be placed.</p> <p>n Is the maximum number of characters to read from input.</p> <p>y Is the y (row) coordinate of starting position of character string to be read.</p> <p>x Is the x (column) coordinate of starting position of character string to be read.</p> <p>win points to the window associated with the terminal from which the character is to be read.</p> |
| DESCRIPTION | <p>The get_wstr() and wget_wstr() functions get a wide character string from the terminal associated with the window <code>stdscr</code> or window <code>win</code>, respectively. The mvget_wstr() and mvwget_wstr() functions move the cursor to the position specified in <code>stdscr</code> or <code>win</code>, respectively, then get a wide character string.</p> <p>These functions call wget_wch(3XC) and place each received character in <code>wstr</code> until a newline character, end-of-line character, or end-of-file character is received, which is also placed in <code>wstr</code>. The erase and kill characters set by the user are processed.</p> <p>The getn_wstr(), mvgetn_wstr(), mvwgetn_wstr() and wgetn_wstr() functions read at most <code>n</code> characters. These functions are used to prevent overflowing the input buffer.</p> |

getn_wstr(3XC)

X/Open Curses Library Functions

RETURN VALUES

On success, these functions return `OK` . Otherwise, they return `ERR` .

ERRORS

None.

SEE ALSO

`get_wch(3XC)` , `getnstr(3XC)`

| | |
|----------------------|---|
| NAME | getopt – get option letter from argument vector |
| SYNOPSIS | <pre>#include <stdlib.h> int getopt(int argc, char * const *argv, const char *optstring); extern char *optarg; extern int optind, opterr, optopt;</pre> |
| DESCRIPTION | <p>The getopt() function returns the next option letter in <i>argv</i> that matches a letter in <i>optstring</i>. It supports all the rules of the command syntax standard (see intro(1)). Since all new commands are intended to adhere to the command syntax standard, they should use getopts(1), getopt(3C) or getsubopt(3C) to parse positional parameters and check for options that are legal for that command.</p> <p>The <i>optstring</i> argument must contain the option letters the command using getopt() will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which may be separated from it by white space. The <i>optarg</i> argument is set to point to the start of the option argument on return from getopt().</p> <p>The getopt() function places in <i>optind</i> the <i>argv</i> index of the next argument to be processed. <i>optind</i> is external and is initialized to 1 before the first call to getopt(). When all options have been processed (that is, up to the first non-option argument), getopt() returns EOF. The special option “—” (two hyphens) may be used to delimit the end of the options; when it is encountered, EOF is returned and “—” is skipped. This is useful in delimiting non-option arguments that begin with “-” (hyphen).</p> |
| RETURN VALUES | <p>The getopt() function prints an error message on the standard error and returns a “?” (question mark) when it encounters an option letter not included in <i>optstring</i> or no argument after an option that expects one. This error message may be disabled by setting <i>opterr</i> to 0. The value of the character that caused the error is in <i>optopt</i>.</p> |
| USAGE | <p>If the application is linked with <code>-lintl</code>, then messages printed from this function are in the native language specified by the LC_MESSAGES locale category; see setlocale(3C).</p> <p>The getopt() function does not fully check for mandatory arguments; that is, given an option string <i>a:b</i> and the input <code>-a -b</code>, getopt() assumes that <code>-b</code> is the mandatory argument to the <code>-a</code> option and not that <code>-a</code> is missing a mandatory argument.</p> <p>It is a violation of the command syntax standard (see intro(1)) for options with arguments to be grouped with other options, as in <code>cmd -abo filename</code>, where <i>a</i> and <i>b</i> are options, <i>o</i> is an option that requires an argument, and</p> |

filename is the argument to `o`. Although this syntax is permitted in the current implementation, it should not be used because it may not be supported in future releases. The correct syntax to use is:

```
cmd -ab -o filename.
```

EXAMPLES

EXAMPLE 1 Example on how one might process the arguments for a command.

The following code fragment shows how one might process the arguments for a command that can take the mutually exclusive options `a` and `b`, and the option `o`, which requires an argument:

```
#include <stdlib.h>
#include <stdio.h>

main (int argc, char **argv)
{
    int c;
    extern char *optarg;
    extern int optind;
    int aflag = 0;
    int bflag = 0;
    int errflag = 0;
    char *ofile = NULL;

    while ((c = getopt(argc, argv, "abo:")) != EOF)
        switch (c) {
            case 'a':
                if (bflag)
                    errflag++;
                else
                    aflag++;
                break;
            case 'b':
                if (aflag)
                    errflag++;
                else
                    bflag++;
                break;
            case 'o':
                ofile = optarg;
                (void)printf("ofile = %s\n", ofile);
                break;
            case '?':
                errflag++;
        }
    if (errflag) {
        (void)fprintf(stderr,
            "usage: cmd [-a|-b] [-o <filename>] files...\n");
        exit (2);
    }
    for ( ; optind < argc; optind++)
        (void)printf("%s\n", argv[optind]);
    return 0;
}
```

}

ATTRIBUTESSee `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO`intro(1)`, `getopts(1)`, `getopt(3C)`, `getsubopt(3C)`, `setlocale(3C)`, `gettext(3C)`, `attributes(5)`

| | |
|----------------------|---|
| NAME | getpagesize – get system page size |
| SYNOPSIS | <pre>#include <unistd.h> int getpagesize(void);</pre> |
| DESCRIPTION | <p>The getpagesize() function returns the number of bytes in a page. Page granularity is the granularity of many of the memory management calls.</p> <p>The page size is a system page size and need not be the same as the underlying hardware page size.</p> <p>The getpagesize() function is equivalent to <code>sysconf (_SC_PAGE_SIZE)</code> and <code>sysconf (_SC_PAGESIZE)</code>. See sysconf(3C).</p> |
| RETURN VALUES | The getpagesize() function returns the current page size. |
| ERRORS | No errors are defined. |
| USAGE | The value returned by getpagesize() need not be the minimum value that malloc(3C) can allocate. Moreover, the application cannot assume that an object of this size can be allocated with malloc() . |
| SEE ALSO | pagesize(1) , brk(2) , getrlimit(2) , mmap(2) , mprotect(2) , munmap(2) , malloc(3C) , msync(3C) , sysconf(3C) |

NAME | `getpass`, `getpassphrase` – read a string of characters without echo

SYNOPSIS | `#include <unistd.h>`

| `char * getpass(const char * prompt);`

| `char * getpassphrase(const char * prompt);`

DESCRIPTION | The `getpass()` function opens the process's controlling terminal, writes to that device the null-terminated string *prompt*, disables echoing, reads a string of characters up to the next newline character or EOF, restores the terminal state and closes the terminal.

| The function `getpassphrase()` is identical to `getpass()`, except that it will read and return a string of up to 256 characters in length.

RETURN VALUES | Upon successful completion, `getpass()` returns a pointer to a null-terminated string of at most `PASS_MAX` bytes that were read from the terminal device. If an error is encountered, the terminal state is restored and a null pointer is returned.

ERRORS | The `getpass()` and `getpassphrase()` functions may fail if:

| **EINTR** | The function was interrupted by a signal.

| **EIO** | The process is a member of a background process attempting to read from its controlling terminal, the process is ignoring or blocking the `SIGTTIN` signal or the process group is orphaned.

| **EMFILE** | `OPEN_MAX` file descriptors are currently open in the calling process.

| **ENFILE** | The maximum allowable number of files is currently open in the system.

| **ENXIO** | The process does not have a controlling terminal.

USAGE | The return value points to static data whose content may be overwritten by each call.

ATTRIBUTES | See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO | [attributes\(5\)](#)

NAME | getpeername – get name of connected peer

SYNOPSIS |

```
cc [ flag ... ] file ... -lsocket -lnsl [ library ... ]
#include <sys/types.h>
#include <sys/socket.h>
```

|

```
int getpeername(int s, struct sockaddr *name, socklen_t *namelen);
```

DESCRIPTION | **getpeername()** returns the name of the peer connected to socket *s*. The *int* pointed to by the *namelen* parameter should be initialized to indicate the amount of space pointed to by *name*. On return it contains the actual size of the name returned (in bytes), prior to any truncation. The name is truncated if the buffer provided is too small.

RETURN VALUES | If successful, **getpeername()** returns 0; otherwise it returns -1 and sets *errno* to indicate the error.

ERRORS | The call succeeds unless:

| | |
|-----------------|--|
| EBADF | The argument <i>s</i> is not a valid descriptor. |
| ENOMEM | There was insufficient user memory for the operation to complete. |
| ENOSR | There were insufficient STREAMS resources available for the operation to complete. |
| ENOTCONN | The socket is not connected. |
| ENOTSOCK | The argument <i>s</i> is not a socket. |

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO | **accept(3N)**, **bind(3N)**, **getsockname(3N)**, **socket(3N)**, **attributes(5)**, **socket(5)**

| | |
|----------------------|---|
| NAME | getpeername – get the name of the peer socket |
| SYNOPSIS | <pre>cc [flag ...] file ... -lxnet [library ...] #include <sys/socket.h></pre> <pre>int getpeername(int socket, struct sockaddr *address, socklen_t *address_len);</pre> |
| DESCRIPTION | <p>The getpeername() function retrieves the peer address of the specified socket, stores this address in the <code>sockaddr</code> structure pointed to by the <i>address</i> argument, and stores the length of this address in the object pointed to by the <i>address_len</i> argument.</p> <p>If the actual length of the address is greater than the length of the supplied <code>sockaddr</code> structure, the stored address will be truncated.</p> <p>If the protocol permits connections by unbound clients, and the peer is not bound, then the value stored in the object pointed to by <i>address</i> is unspecified.</p> |
| RETURN VALUES | Upon successful completion, 0 is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error. |
| ERRORS | <p>The getpeername() function will fail if:</p> <p>EBADF The <i>socket</i> argument is not a valid file descriptor.</p> <p>EFAULT The <i>address</i> or <i>address_len</i> parameter can not be accessed or written.</p> <p>EINVAL The socket has been shut down.</p> <p>ENOTCONN The socket is not connected or otherwise has not had the peer prespecified.</p> <p>ENOTSOCK The <i>socket</i> argument does not refer to a socket.</p> <p>EOPNOTSUPP The operation is not supported for the socket protocol.</p> <p>The getpeername() function may fail if:</p> <p>ENOBUFS Insufficient resources were available in the system to complete the call.</p> <p>ENOSR There were insufficient STREAMS resources available for the operation to complete.</p> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`accept(3XN)`, `bind(3XN)`, `getsockname(3XN)`, `socket(3XN)`,
`attributes(5)`

| | |
|----------------------|--|
| NAME | getpriority, setpriority – get or set process scheduling priority |
| SYNOPSIS | <pre>#include <sys/resource.h> int getpriority(int which, id_t who); int setpriority(int which, id_t who, int priority);</pre> |
| DESCRIPTION | <p>The getpriority() function obtains the current scheduling priority of a process, process group, or user. The setpriority() function sets the scheduling priority of a process, process group, or user.</p> <p>Target processes are specified by the values of the <i>which</i> and <i>who</i> arguments. The <i>which</i> argument may be one of the following values: <code>PRIO_PROCESS</code>, <code>PRIO_PGRP</code>, or <code>PRIO_USER</code>, indicating that the <i>who</i> argument is to be interpreted as a process ID, a process group ID, or a user ID, respectively. A 0 value for the <i>who</i> argument specifies the current process, process group, or user.</p> <p>If more than one process is specified, getpriority() returns the highest priority (lowest numerical value) pertaining to any of the specified processes, and setpriority() sets the priorities of all of the specified processes to the specified value.</p> <p>The default <i>priority</i> is 0; negative priorities cause more favorable scheduling. While the range of valid priority values is [-20, 20], implementations may enforce more restrictive limits. If the value specified to setpriority() is less than the system's lowest supported priority value, the system's lowest supported value is used; if it is greater than the system's highest supported value, the system's highest supported value is used.</p> <p>Only a process with appropriate privileges can raise its priority (that is, assign a lower numerical priority value).</p> |
| RETURN VALUES | <p>Upon successful completion, getpriority() returns an integer in the range from -20 to 20. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error.</p> <p>Upon successful completion, setpriority() returns 0. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error.</p> |
| ERRORS | <p>The getpriority() and setpriority() functions will fail if:</p> <p>ESRCH No process could be located using the <i>which</i> and <i>who</i> argument values specified.</p> <p>EINVAL The value of the <i>which</i> argument was not recognized, or the value of the <i>who</i> argument is not a valid process ID, process group ID, or user ID.</p> <p>In addition, setpriority() may fail if:</p> |

EPERM A process was located, but neither the real nor effective user ID of the executing process is the privileged user or match the effective user ID of the process whose priority is being changed.

EACCES A request was made to change the priority to a lower numeric value (that is, to a higher priority) and the current process does not have appropriate privileges.

USAGE The effect of changing the scheduling priority may vary depending on the process-scheduling algorithm in effect.

Because **getpriority()** can return `-1` on successful completion, it is necessary to set `errno` to `0` prior to a call to **getpriority()**. If **getpriority()** returns `-1`, then `errno` can be checked to see if an error occurred or if the value is a legitimate priority.

SEE ALSO `nice(1)`, `renice(1)`, `fork(2)`

| | |
|--------------------|--|
| NAME | getprotobyname, getprotobyname_r, getprotobynumber, getprotobynumber_r, getprotoent, getprotoent_r, setprotoent, endprotoent – get protocol entry |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lsocket -lnsl [<i>library</i> ...] #include <netdb.h> struct protoent * getprotobyname(const char * <i>name</i>); struct protoent * getprotobyname_r(const char * <i>name</i>, struct protoent * <i>result</i>, char * <i>buffer</i>, int <i>buflen</i>); struct protoent * getprotobynumber(int <i>proto</i>); struct protoent * getprotobynumber_r(int <i>proto</i>, struct protoent * <i>result</i>, char * <i>buffer</i>, int <i>buflen</i>); struct protoent * getprotoent(void); struct protoent * getprotoent_r(struct protoent * <i>result</i>, char * <i>buffer</i>, int <i>buflen</i>); int setprotoent(int <i>stayopen</i>); int endprotoent(void); </pre> |
| DESCRIPTION | <p>These routines return a protocol entry. Two types of interfaces are supported: reentrant (getprotobyname_r() , getprotobynumber_r() , and getprotoent_r()) and non-reentrant (getprotobyname() , getprotobynumber() , and getprotoent()). The reentrant routines may be used in single-threaded applications and are safe for multi-threaded applications, making them the preferred interfaces.</p> <p>The reentrant routines require additional parameters which are used to return results data. <i>result</i> is a pointer to a <code>struct protoent</code> structure and will be where the returned results will be stored. <i>buffer</i> is used as storage space for elements of the returned results. <i>buflen</i> is the size of <i>buffer</i> and should be large enough to contain all returned data. <i>buflen</i> must be at least 1024 bytes.</p> |

getprotobyname_r() , **getprotobynumber_r()** , and **getprotoent_r()** each return a protocol entry.

The entry may come from one of the following sources: the protocols file (see **protocols(4)**), the NIS maps “protocols.byname” and “protocols.bynumber”, and the NIS+ table “protocols”. The sources and their lookup order are specified in the `/etc/nsswitch.conf` file (see **nsswitch.conf(4)** for details). Some name services such as NIS will return only one name for a host, whereas others such as NIS+ or DNS will return all aliases.

getprotobyname_r() and **getprotobynumber_r()** sequentially search from the beginning of the file until a matching protocol name or protocol number is found, or until an EOF is encountered.

getprotobyname() and **getprotobynumber()** have the same functionality as **getprotobyname_r()** and **getprotobynumber_r()** except that a static buffer is used to store returned results. These routines are unsafe in a multi-threaded application.

getprotoent_r() enumerates protocol entries: successive calls to **getprotoent_r()** will return either successive protocol entries or NULL. Enumeration may not be supported by some sources. Note that if multiple threads call **getprotoent_r()** , each will retrieve a subset of the protocol database.

getprotent() has the same functionality as **getprotent_r()** except that a static buffer is used to store returned results. This routine is unsafe in a multi-threaded application.

setprotoent() “rewinds” to the beginning of the enumeration of protocol entries. If the *stayopen* flag is non-zero, resources such as open file descriptors are not deallocated after each call to **getprotobynumber_r()** and **getprotobyname_r()** . Calls to **getprotobyname_r()** , **getprotobyname()** , **getprotobynumber_r()** and **getprotobynumber()** may leave the enumeration in an indeterminate state, so **setprotoent()** should be called before the first **getprotoent_r()** or **getprotoent()** . Note that **setprotoent()** has process-wide scope, and “rewinds” the protocol entries for all threads calling **getprotoent_r()** as well as main-thread calls to **getprotoent()** .

endprotoent() may be called to indicate that protocol processing is complete; the system may then close any open protocols file, deallocate storage, and so forth. It is legitimate, but possibly less efficient, to call more protocol routines after **endprotoent()** .

The internal representation of a protocol entry is a `protoent` structure defined in `<netdb.h>` with the following members:

```
c
  char  *p_name;
```

```
char  **p_aliases;
int    p_proto;
```

RETURN VALUES

`getprotobyname_r()`, `getprotobyname()`, `getprotobynumber_r()`, and **getprotobynumber()** return a pointer to a struct `protoent` if they successfully locate the requested entry; otherwise they return `NULL`.

getprotoent_r() and **getprotoent()** return a pointer to a struct `protoent` if they successfully enumerate an entry; otherwise they return `NULL`, indicating the end of the enumeration.

ERRORS

`getprotobyname_r()`, `getprotobynumber_r()`, and **getprotoent_r()** will fail if the following is true:

ERANGE length of the buffer supplied by caller is not large enough to store the result.

FILES

`/etc/protocols`
`/etc/nsswitch.conf`

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | See NOTES below. |

SEE ALSO

intro(3), **nsswitch.conf(4)**, **protocols(4)**, **attributes(5)**, **netdb(5)**

NOTES

Although **getprotobyname_r()**, **getprotobynumber_r()**, and **getprotoent_r()** are not mentioned by POSIX.4a Draft 6, they were added to complete the functionality provided by similar thread-safe functions. These interfaces are subject to change to be compatible with the "spirit" of POSIX.4a when it is approved as a standard.

When compiling multithreaded applications, see **intro(3)**, *Notes On Multithread Applications*, for information about the use of the `_REENTRANT` flag.

The routines **getprotobyname_r()**, **getprotobynumber_r()**, and **getprotoent_r()** are reentrant and multi-thread safe. The reentrant interfaces can be used in single-threaded as well as multi-threaded applications and are therefore the preferred interfaces.

The routines **getprotobyname()** , **getprotobyaddr()** , and **getprotoent()** use static storage, so returned data must be copied if it is to be saved. Because of their use of static storage for returned data, these routines are not safe for multi-threaded applications.

setprotoent() and **endprotoent()** have process-wide scope, and are therefore not safe in multi-threaded applications.

Use of `getprotoent_r()` and `getprotoent()` is discouraged; enumeration is well-defined for the protocols file and is supported (albeit inefficiently) for NIS and NIS+, but in general may not be well-defined. The semantics of enumeration are discussed in `nsswitch.conf(4)` .

BUGS

Only the Internet protocols are currently understood.

Programs that call **getprotobyname_r()** or **getprotobynumber_r()** routines cannot be linked statically since the implementation of these routines requires dynamic linker functionality to access shared objects at run time.

| NAME | getpublickey, getsecretkey, publickey – retrieve public or secret key | | | | |
|----------------------|--|----------------|-----------------|----------|------|
| SYNOPSIS | <pre>#include <rpc/rpc.h> #include <rpc/key_prot.h> int getpublickey(const char netname[MAXNETNAMELEN], char publickey[HEXKEYBYTES+1]); int getsecretkey(const char netname[MAXNETNAMELEN], char secretkey[HEXKEYBYTES+1], const char * passwd);</pre> | | | | |
| DESCRIPTION | <p>getpublickey() and getsecretkey() get public and secret keys for <i>netname</i> . The key may come from one of the following sources: the <code>/etc/publickey</code> file (see publickey(4)) or the NIS map “publickey.byname” or the NIS+ table “cred.org_dir”. The sources and their lookup order are specified in the <code>/etc/nsswitch.conf</code> file (see nsswitch.conf(4)).</p> <p>getsecretkey() has an extra argument, <code>passwd</code> , used to decrypt the encrypted secret key stored in the database.</p> | | | | |
| RETURN VALUES | Both routines return 1 if they are successful in finding the key, 0 otherwise. The keys are returned as NULL-terminated, hexadecimal strings. If the password supplied to getsecretkey() fails to decrypt the secret key, the routine will return 1 but the <code>secretkey [0]</code> will be set to NULL. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe | | | | |
| SEE ALSO | secure_rpc(3N) , nsswitch.conf(4) , publickey(4) , attributes(5) | | | | |
| WARNINGS | If getpublickey() gets the public key from any source other than NIS+, all authenticated NIS+ operations may fail. To ensure that this does not happen, edit the nsswitch.conf(4) file to make sure that the public key is obtained from NIS+. | | | | |

| NAME | getpw – get passwd entry from UID | | | | |
|----------------------|--|----------------|-----------------|----------|------|
| SYNOPSIS | <pre>#include <stdlib.h> int getpw(uid_t uid, char *buf);</pre> | | | | |
| DESCRIPTION | The getpw() function searches the user data base for a user id number that equals <i>uid</i> , copies the line of the password file in which <i>uid</i> was found into the array pointed to by <i>buf</i> , and returns 0. getpw() returns non-zero if <i>uid</i> cannot be found. | | | | |
| USAGE | <p>This function is included only for compatibility with prior systems and should not be used; the functions described on the getpwnam(3C) manual page should be used instead.</p> <p>If the <code>/etc/passwd</code> and the <code>/etc/group</code> files have a plus sign (+) for the NIS entry, then getpwent() and getgrent() will not return NULL when the end of file is reached. See getpwnam(3C).</p> | | | | |
| RETURN VALUES | The getpw() function returns non-zero on error. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe | | | | |
| SEE ALSO | getpwnam(3C) , passwd(4) , attributes(5) | | | | |

| | |
|--------------------|---|
| NAME | getpwnam, getpwnam_r, getpwent, getpwent_r, getpwuid, getpwuid_r, setpwent, endpwent, fgetpwent, fgetpwent_r – get password entry |
| SYNOPSIS | <pre>#include <pwd.h> struct passwd * getpwnam(const char * <i>name</i>); struct passwd * getpwnam_r(const char * <i>name</i>, struct passwd * <i>pwd</i>, char * <i>buffer</i>, int <i>buflen</i>); struct passwd * getpwent(void); struct passwd * getpwent_r(struct passwd * <i>pwd</i>, char * <i>buffer</i>, int <i>buflen</i>); struct passwd * getpwuid(uid_t <i>uid</i>); struct passwd * getpwuid_r(uid_t <i>uid</i>, struct passwd * <i>pwd</i>, char * <i>buffer</i>, int <i>buflen</i>); void setpwent(void); void endpwent(void); struct passwd * fgetpwent(FILE * <i>f</i>); struct passwd * fgetpwent_r(FILE * <i>f</i>, struct passwd * <i>pwd</i>, char * <i>buffer</i>, int <i>buflen</i>);</pre> |
| POSIX | <pre>cc [<i>flag</i> ...] <i>file</i> ... -D_POSIX_PTHREAD_SEMANTICS [<i>library</i> ...] int getpwnam_r(const char * <i>name</i>, struct passwd * <i>pwd</i>, char * <i>buffer</i>, size_t <i>bufsize</i>, struct passwd ** <i>result</i>); int getpwuid_r(uid_t <i>uid</i>, struct passwd * <i>pwd</i>, char * <i>buffer</i>, size_t <i>bufsize</i>, struct passwd ** <i>result</i>);</pre> |
| DESCRIPTION | <p>These functions are used to obtain password entries. Entries can come from any of the sources for <code>passwd</code> specified in the <code>/etc/nsswitch.conf</code> file (see <code>nsswitch.conf(4)</code>).</p> <p>The <code>getpwnam()</code> function searches for a password entry with the login name specified by the character string parameter <code>name</code>.</p> <p>The <code>getpwuid()</code> function searches for a password entry with the (numeric) user ID specified by the parameter <code>uid</code>.</p> |

The **setpwent()**, **getpwent()**, and **endpwent()** functions are used to enumerate password entries from the database. **setpwent()** sets (or resets) the enumeration to the beginning of the set of password entries. This function should be called before the first call to **getpwent()**. Calls to **getpwnam()** and **getpwuid()** leave the enumeration position in an indeterminate state. Successive calls to **getpwent()** return either successive entries or `NULL`, indicating the end of the enumeration.

The **endpwent()** function may be called to indicate that the caller expects to do no further password retrieval operations; the system may then close the password file, deallocate resources it was using, and so forth. It is still allowed, but possibly less efficient, for the process to call more password functions after calling **endpwent()**.

The **fgetpwent()** function, unlike the other functions above, does not use `nsswitch.conf`; it reads and parses the next line from the stream *f*, which is assumed to have the format of the `passwd` file. See **passwd(4)**.

Reentrant Interfaces

The functions **getpwnam()**, **getpwuid()**, **getpwent()**, and **fgetpwent()** use static storage that is re-used in each call, making these routines unsafe for use in multithreaded applications.

The parallel functions **getpwnam_r()**, **getpwuid_r()**, **getpwent_r()**, and **fgetpwent_r()** provide reentrant interfaces for these operations.

Each reentrant interface performs the same operation as its non-reentrant counterpart, named by removing the “_r” suffix. The reentrant interfaces, however, use buffers supplied by the caller to store returned results, and are safe for use in both single-threaded and multithreaded applications.

Each reentrant interface takes the same parameters as its non-reentrant counterpart, as well as the following additional parameters. The parameter *pwd* must be a pointer to a `struct passwd` structure allocated by the caller. On successful completion, the function returns the password entry in this structure. The parameter *buffer* is a pointer to a buffer supplied by the caller, used as storage space for the password data. All of the pointers within the returned `struct passwd pwd` point to data stored within this buffer; see **RETURN VALUES**. The buffer must be large enough to hold all the data associated with the password entry. The parameter *buflen* (or *bufsize* for the POSIX versions; see **standards(5)**) should give the size in bytes of *buffer*. The POSIX versions place a pointer to the modified `pwd` structure in the *result* parameter, instead of returning a pointer to this structure.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. The **setpwent()** function may be used in a multithreaded application but resets the enumeration

position for all threads. If multiple threads interleave calls to `getpwent_r()`, the threads will enumerate disjoint subsets of the password database.

Like their non-reentrant counterparts, `getpwnam_r()` and `getpwuid_r()` leave the enumeration position in an indeterminate state.

RETURN VALUES

Password entries are represented by the `struct passwd` structure defined in `<pwd.h>`:

```
struct passwd {
    char *pw_name;          /* user's login name */
    char *pw_passwd;       /* no longer used */
    uid_t pw_uid;          /* user's uid */
    gid_t pw_gid;          /* user's gid */
    char *pw_age;          /* not used */
    char *pw_comment;      /* not used */
    char *pw_gecos;        /* typically user's full name */
    char *pw_dir;          /* user's home dir */
    char *pw_shell;        /* user's login shell */
};
```

The `pw_passwd` member should not be used as the encrypted password for the user; use `getspnam()` or `getspnam_r()` instead. See `getspnam(3C)`.

The `getpwnam()`, `getpwnam_r()`, `getpwuid()`, and `getpwuid_r()` functions each return a pointer to a `struct passwd` if they successfully locate the requested entry; otherwise they return `NULL`. The POSIX functions `getpwnam_r()` and `getpwuid_r()` return 0 upon success, or the error number in case of failure.

The `getpwent()`, `getpwent_r()`, `fgetpwent()`, and `fgetpwent_r()` functions each return a pointer to a `struct passwd` if they successfully enumerate an entry; otherwise they return `NULL`, indicating the end of the enumeration.

The `getpwnam()`, `getpwuid()`, `getpwent()`, and `fgetpwent()` functions use static storage, so returned data must be copied before a subsequent call to any of these functions if the data is to be saved.

When the pointer returned by the reentrant functions `getpwnam_r()`, `getpwuid_r()`, `getpwent_r()`, and `fgetpwent_r()` is non-null, it is always equal to the `pwd` pointer that was supplied by the caller.

ERRORS

The reentrant functions `getpwnam_r()`, `getpwuid_r()`, `getpwent_r()`, and `fgetpwent_r()` will return `NULL` and set `errno` to `ERANGE` (or in the case of POSIX functions `getpwnam_r()` and `getpwuid_r()` return the `ERANGE` error) if the length of the buffer supplied by caller is not large enough to store the result. See `Intro(2)` for the proper usage and interpretation of `errno` in multithreaded applications.

USAGE Applications that use the interfaces described on this manual page cannot be linked statically, since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|---|
| MT-Level | See "Reentrant Interfaces" in DESCRIPTION . |

SEE ALSO **nispasswd(1)** , **passwd(1)** , **yppasswd(1)** , **Intro(2)** , **Intro(3)** , **cuserid(3S)** , **getgrnam(3C)** , **getlogin(3C)** , **getspnam(3C)** , **nsswitch.conf(4)** , **passwd(4)** , **shadow(4)** , **attributes(5)** , **standards(5)**

NOTES When compiling multithreaded programs, see **Intro(3)** , *Notes On Multithreaded Applications* .

Use of the enumeration interfaces **getpwent()** and **getpwent_r()** is discouraged; enumeration is supported for the `passwd` file, NIS, and NIS+, but in general is not efficient and may not be supported for all database sources. The semantics of enumeration are discussed further in **nsswitch.conf(4)** .

Previous releases allowed the use of '+' and '-' entries in `/etc/passwd` to selectively include and exclude NIS entries. The primary usage of these '+/-' entries is superseded by the name service switch, so the '+/-' form may not be supported in future releases.

If required, the '+/-' functionality can still be obtained for NIS by specifying `compat` as the source for `passwd` .

If the '+/-' functionality is required in conjunction with NIS+, specify both `compat` as the source for `passwd` and `nisplus` as the source for the pseudo-database `passwd_compat` . See **passwd(4)** , **shadow(4)** , and **nsswitch.conf(4)** for details.

If the '+/-' is used, both `/etc/shadow` and `/etc/passwd` should have the same '+' and '-' entries to ensure consistency between the password and shadow databases.

If a password entry from any of the sources contains an empty `uid` or `gid` field, that entry will be ignored by the files, NIS , and NIS+ name service switch backends. This will cause the user to appear unknown to the system.

If a password entry contains an empty *gecos* , *home directory* , or *shell* field, **getpwnam()** and **getpwnam_r()** return a pointer to a null string in the respective field of the `passwd` structure.

If the shell field is empty, **login(1)** automatically assigns the default shell. See **login(1)** .

Solaris 2.4 and earlier releases provided definitions of the **getpwnam_r()** and **getpwuid_r()** functions as specified in POSIX.1c Draft 6. The final POSIX.1c standard changed the interface for these functions. Support for the Draft 6 interface is provided for compatibility only and may not be supported in future releases. New applications and libraries should use the POSIX standard interface.

For POSIX.1c-compliant applications, the `_POSIX_PTHREAD_SEMANTICS` and `_REENTRANT` flags are automatically turned on by defining the `_POSIX_C_SOURCE` flag with a value $\geq 199506L$.

| | |
|--------------------|--|
| NAME | getrpcbyname, getrpcbyname_r, getrpcbynumber, getrpcbynumber_r, getrpccent, getrpccent_r, setrpccent, endrpccent – get RPC entry |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] #include <rpc/rpcent.h> struct rpcent * getrpcbyname(const char * <i>name</i>); struct rpcent * getrpcbyname_r(const char * <i>name</i>, struct rpcent * <i>result</i>, char * <i>buffer</i>, int <i>buflen</i>); struct rpcent * getrpcbynumber(const int <i>number</i>); struct rpcent * getrpcbynumber_r(const int <i>number</i>, struct rpcent * <i>result</i>, char * <i>buffer</i>, int <i>buflen</i>); struct rpcent * getrpccent(void); struct rpcent * getrpccent_r(struct rpcent * <i>result</i>, char * <i>buffer</i>, int <i>buflen</i>); void setrpccent(const int <i>stayopen</i>); void endrpccent(void);</pre> |
| DESCRIPTION | <p>These functions are used to obtain entries for RPC (Remote Procedure Call) services. An entry may come from any of the sources for <code>rpc</code> specified in the <code>/etc/nsswitch.conf</code> file (see <code>nsswitch.conf(4)</code>).</p> <p>getrpcbyname() searches for an entry with the RPC service name specified by the parameter <i>name</i>.</p> <p>getrpcbynumber() searches for an entry with the RPC program number <i>number</i>.</p> <p>The functions setrpccent(), getrpccent(), and endrpccent() are used to enumerate RPC entries from the database.</p> <p>setrpccent() sets (or resets) the enumeration to the beginning of the set of RPC entries. This function should be called before the first call to getrpccent(). Calls to getrpcbyname() and getrpcbynumber() leave the enumeration position in</p> |

an indeterminate state. If the *stayopen* flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to **endrpcnt()** .

Successive calls to **getrpcnt()** return either successive entries or NULL, indicating the end of the enumeration.

endrpcnt() may be called to indicate that the caller expects to do no further RPC entry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more RPC entry retrieval functions after calling **endrpcnt()** .

Reentrant Interfaces

The functions **getrpcbyname()** , **getrpcbynumber()** , and **getrpcnt()** use static storage that is re-used in each call, making these routines unsafe for use in multithreaded applications.

The functions **getrpcbyname_r()** , **getrpcbynumber_r()** , and **getrpcnt_r()** provide reentrant interfaces for these operations.

Each reentrant interface performs the same operation as its non-reentrant counterpart, named by removing the “_r” suffix. The reentrant interfaces, however, use buffers supplied by the caller to store returned results, and are safe for use in both single-threaded and multithreaded applications.

Each reentrant interface takes the same parameters as its non-reentrant counterpart, as well as the following additional parameters. The parameter *result* must be a pointer to a `struct rpcnt` structure allocated by the caller. On successful completion, the function returns the RPC entry in this structure. The parameter *buffer* must be a pointer to a buffer supplied by the caller. This buffer is used as storage space for the RPC entry data. All of the pointers within the returned `struct rpcnt result` point to data stored within this buffer (see RETURN VALUES). The buffer must be large enough to hold all of the data associated with the RPC entry. The parameter *buflen* should give the size in bytes of the buffer indicated by *buffer* .

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. **setrpcnt()** may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to **getrpcnt_r()** , the threads will enumerate disjoint subsets of the RPC entry database.

Like their non-reentrant counterparts, **getrpcbyname_r()** and **getrpcbynumber_r()** leave the enumeration position in an indeterminate state.

RETURN VALUES

RPC entries are represented by the `struct rpcnt` structure defined in `<rpc/rpcnt.h>` :

```
struct rpcnt {
    char *r_name;          /* name of this rpc service
```

```

    char **r_aliases; /* zero-terminated list of alternate names */
    int r_number;     /* rpc program number */
};

```

The functions **getrpcbyname()**, **getrpcbyname_r()**, **getrpcbynumber()**, and **getrpcbynumber_r()** each return a pointer to a `struct rpcent` if they successfully locate the requested entry; otherwise they return `NULL`.

The functions **getrpcent()** and **getrpcent_r()** each return a pointer to a `struct rpcent` if they successfully enumerate an entry; otherwise they return `NULL`, indicating the end of the enumeration.

The functions **getrpcbyname()**, **getrpcbynumber()**, and **getrpcent()** use static storage, so returned data must be copied before a subsequent call to any of these functions if the data is to be saved.

When the pointer returned by the reentrant functions **getrpcbyname_r()**, **getrpcbynumber_r()**, and **getrpcent_r()** is non-`NULL`, it is always equal to the *result* pointer that was supplied by the caller.

ERRORS

The reentrant functions **getrpcbyname_r()**, **getrpcbynumber_r()** and **getrpcent_r()** will return `NULL` and set `errno` to `ERANGE` if the length of the buffer supplied by caller is not large enough to store the result. See **intro(2)** for the proper usage and interpretation of `errno` in multithreaded applications.

FILES

```

/etc/rpc
/etc/nsswitch.conf

```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|---|
| MT-Level | See "Reentrant Interfaces" in DESCRIPTION . |

SEE ALSO

rpcinfo(1M), **rpc(3N)**, **nsswitch.conf(4)**, **rpc(4)**, **attributes(5)**

WARNINGS

The reentrant interfaces **getrpcbyname_r()**, **getrpcbynumber_r()**, and **getrpcent_r()** are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.

NOTES

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see `intro(3)`, *Notes On Multithreaded Applications*, for information about the use of the `_REENTRANT` flag.

Use of the enumeration interfaces `getrpcent()` and `getrpcent_r()` is discouraged; enumeration may not be supported for all database sources. The semantics of enumeration are discussed further in `nsswitch.conf(4)`.

| | | | | | | | | | |
|------------------------|--|-----------------------|---|-----------------------|---|------------------------|--|-----------------------|---|
| NAME | getrusage – get information about resource utilization | | | | | | | | |
| SYNOPSIS | <pre>#include <sys/resource.h> int getrusage(int who, struct rusage *r_usage);</pre> | | | | | | | | |
| DESCRIPTION | <p>The getrusage() function provides measures of the resources used by the current process or its terminated and waited-for child processes. If the value of the <code>who</code> argument is <code>RUSAGE_SELF</code>, information is returned about resources used by the current process. If the value of the <code>who</code> argument is <code>RUSAGE_CHILDREN</code>, information is returned about resources used by the terminated and waited-for children of the current process. If the child is never waited for (for instance, if the parent has <code>SA_NOCLDWAIT</code> set or sets <code>SIGCHLD</code> to <code>SIG_IGN</code>), the resource information for the child process is discarded and not included in the resource information provided by getrusage().</p> <p>The <code>r_usage</code> argument is a pointer to an object of type <code>struct rusage</code> in which the returned information is stored. The members of <code>rusage</code> are as follows:</p> <pre>struct timeval ru_utime; /* user time used */ struct timeval ru_stime; /* system time used */ long ru_maxrss; /* maximum resident set size */ long ru_idrss; /* integral resident set size */ long ru_minflt; /* page faults not requiring physical I/O */ long ru_majflt; /* page faults requiring physical I/O */ long ru_nswap; /* swaps */ long ru_inblock; /* block input operations */ long ru_oublock; /* block output operations */ long ru_msgsnd; /* messages sent */ long ru_msrvcv; /* messages received */ long ru_nsignals; /* signals received */ long ru_nvcsw; /* voluntary context switches */ long ru_nivcsw; /* involuntary context switches */</pre> <p>The structure members are interpreted as follows:</p> <table border="0"> <tr> <td style="padding-right: 20px;"><code>ru_utime</code></td> <td>The total amount of time spent executing in user mode. Time is given in seconds and microseconds.</td> </tr> <tr> <td style="padding-right: 20px;"><code>ru_stime</code></td> <td>The total amount of time spent executing in system mode. Time is given in seconds and microseconds.</td> </tr> <tr> <td style="padding-right: 20px;"><code>ru_maxrss</code></td> <td>The maximum resident set size. Size is given in pages (the size of a page, in bytes, is given by the getpagesize(3C) function). See the NOTES section of this page.</td> </tr> <tr> <td style="padding-right: 20px;"><code>ru_idrss</code></td> <td>An “integral” value indicating the amount of memory in use by a process while the process is running. This value is the sum of the resident set sizes of the process running when a</td> </tr> </table> | <code>ru_utime</code> | The total amount of time spent executing in user mode. Time is given in seconds and microseconds. | <code>ru_stime</code> | The total amount of time spent executing in system mode. Time is given in seconds and microseconds. | <code>ru_maxrss</code> | The maximum resident set size. Size is given in pages (the size of a page, in bytes, is given by the getpagesize(3C) function). See the NOTES section of this page. | <code>ru_idrss</code> | An “integral” value indicating the amount of memory in use by a process while the process is running. This value is the sum of the resident set sizes of the process running when a |
| <code>ru_utime</code> | The total amount of time spent executing in user mode. Time is given in seconds and microseconds. | | | | | | | | |
| <code>ru_stime</code> | The total amount of time spent executing in system mode. Time is given in seconds and microseconds. | | | | | | | | |
| <code>ru_maxrss</code> | The maximum resident set size. Size is given in pages (the size of a page, in bytes, is given by the getpagesize(3C) function). See the NOTES section of this page. | | | | | | | | |
| <code>ru_idrss</code> | An “integral” value indicating the amount of memory in use by a process while the process is running. This value is the sum of the resident set sizes of the process running when a | | | | | | | | |

| | |
|--------------------------|---|
| | clock tick occurs. The value is given in pages times clock ticks. It does not take sharing into account. See the NOTES section of this page. |
| <code>ru_minflt</code> | The number of page faults serviced which did not require any physical I/O activity. See the NOTES section of this page. |
| <code>ru_majflt</code> | The number of page faults serviced which required physical I/O activity. This could include page ahead operations by the kernel. See the NOTES section of this page. |
| <code>ru_nswap</code> | The number of times a process was swapped out of main memory. |
| <code>ru_inblock</code> | The number of times the file system had to perform input in servicing a <code>read(2)</code> request. |
| <code>ru_oublock</code> | The number of times the file system had to perform output in servicing a <code>write(2)</code> request. |
| <code>ru_msgsnd</code> | The number of messages sent over sockets. |
| <code>ru_msgrcv</code> | The number of messages received from sockets. |
| <code>ru_nsignals</code> | The number of signals delivered. |
| <code>ru_nvcsw</code> | The number of times a context switch resulted due to a process voluntarily giving up the processor before its time slice was completed (usually to await availability of a resource). |
| <code>ru_nivcsw</code> | The number of times a context switch resulted due to a higher priority process becoming runnable or because the current process exceeded its time slice. |

RETURN VALUES

Upon successful completion, `getrusage()` returns 0. Otherwise, -1 is returned and `errno` is set to indicate the error.

ERRORS

The `getrusage()` function will fail if:

| | |
|---------------|---|
| EFAULT | The address specified by the <code>r_usage</code> argument is not in a valid portion of the process' address space. |
| EINVAL | The <code>who</code> parameter is not a valid value. |

SEE ALSO | `sar(1M)`, `read(2)`, `times(2)`, `wait(2)`, `write(2)`, `getpagesize(3C)`,
`gettimeofday(3C)`

NOTES | Only the `timeval` member of `struct rusage` are supported in this implementation.

The numbers `ru_inblock` and `ru_oublock` account only for real I/O, and are approximate measures at best. Data supplied by the cache mechanism is charged only to the first process to read and the last process to write the data.

The way resident set size is calculated is an approximation, and could misrepresent the true resident set size.

Page faults can be generated from a variety of sources and for a variety of reasons. The customary cause for a page fault is a direct reference by the program to a page which is not in memory. Now, however, the kernel can generate page faults on behalf of the user, for example, servicing `read(2)` and `write(2)` functions. Also, a page fault can be caused by an absent hardware translation to a page, even though the page is in physical memory.

In addition to hardware detected page faults, the kernel may cause pseudo page faults in order to perform some housekeeping. For example, the kernel may generate page faults, even if the pages exist in physical memory, in order to lock down pages involved in a raw I/O request.

By definition, major page faults require physical I/O, while minor page faults do not require physical I/O. For example, reclaiming the page from the free list would avoid I/O and generate a minor page fault. More commonly, minor page faults occur during process startup as references to pages which are already in memory. For example, if an address space faults on some "hot" executable or shared library, this results in a minor page fault for the address space. Also, any one doing a `read(2)` or `write(2)` to something that is in the page cache will get a minor page fault(s) as well.

There is no way to obtain information about a child process which has not yet terminated.

| NAME | gets, fgets – get a string from a stream | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <stdio.h> char * gets(char * s); char * fgets(char * s, int n, FILE * stream);</pre> | | | | |
| DESCRIPTION | <p>The gets() function reads characters from the standard input stream (see intro(3)), <code>stdin</code>, into the array pointed to by <code>s</code>, until a newline character is read or an end-of-file condition is encountered. The newline character is discarded and the string is terminated with a null character.</p> <p>The fgets() function reads characters from the <i>stream</i> into the array pointed to by <code>s</code>, until <code>n - 1</code> characters are read, or a newline character is read and transferred to <code>s</code>, or an end-of-file condition is encountered. The string is then terminated with a null character.</p> <p>When using gets(), if the length of an input line exceeds the size of <code>s</code>, indeterminate behavior may result. For this reason, it is strongly recommended that gets() be avoided in favor of fgets().</p> | | | | |
| RETURN VALUES | <p>If end-of-file is encountered and no characters have been read, no characters are transferred to <code>s</code> and a null pointer is returned. If a read error occurs, such as trying to use these functions on a file that has not been opened for reading, a null pointer is returned and the error indicator for the stream is set. If end-of-file is encountered, the EOF indicator for the stream is set. Otherwise <code>s</code> is returned.</p> | | | | |
| ERRORS | <p>The gets() and fgets() functions will fail if data needs to be read and:</p> <p>EOverflow The file is a regular file and an attempt was made to read at or beyond the offset maximum associated with the corresponding <i>stream</i>.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | <p>lseek(2), read(2), ferror(3S), fopen(3S), fread(3S), getc(3S), scanf(3S), stdio(3S), ungetc(3S), attributes(5)</p> | | | | |

| | |
|--------------------|---|
| NAME | getservbyname, getservbyname_r, getservbyport, getservbyport_r, getservent, getservent_r, setservent, endservent – get service entry |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lsocket -lnsl [<i>library</i> ...] #include <netdb.h> struct servent * getservbyname(const char * <i>name</i>, const char * <i>proto</i>); struct servent * getservbyname_r(const char * <i>name</i>, const char * <i>proto</i>, struct servent * <i>result</i>, char * <i>buffer</i>, int <i>buflen</i>); struct servent * getservbyport(int <i>port</i>, const char * <i>proto</i>); struct servent * getservbyport_r(int <i>port</i>, const char * <i>proto</i>, struct servent * <i>result</i>, char * <i>buffer</i>, int <i>buflen</i>); struct servent * getservent(void); struct servent * getservent_r(struct servent * <i>result</i>, char * <i>buffer</i>, int <i>buflen</i>); int setservent(int <i>stayopen</i>); int endservent(void);</pre> |
| DESCRIPTION | <p>These functions are used to obtain entries for Internet services. An entry may come from any of the sources for <i>services</i> specified in the <i>/etc/nsswitch.conf</i> file. See nsswitch.conf(4) .</p> <p>getservbyname() and getservbyport() sequentially search from the beginning of the file until a matching protocol name or port number is found, or until end-of-file is encountered. If a protocol name is also supplied (non- NULL), searches must also match the protocol.</p> <p>getservbyname() searches for an entry with the Internet service name specified by the parameter <i>name</i> .</p> <p>getservbyport() searches for an entry with the Internet port number <i>port</i> .</p> |

All addresses are returned in network order. In order to interpret the addresses, `byteorder(3N)`

must be used for byte order conversion. The string *proto* is used by both `getservbyname()` and `getservbyport()` to restrict the search to entries with the specified protocol. If *proto* is `NULL`, entries with any protocol may be returned.

The functions `setservent()`, `getservent()`, and `endservent()` are used to enumerate entries from the services database.

`setservent()` sets (or resets) the enumeration to the beginning of the set of service entries. This function should be called before the first call to `getservent()`. Calls to the functions `getservbyname()` and `getservbyport()` leave the enumeration position in an indeterminate state. If the *stayopen* flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to `endservent()`.

`getservent()` reads the next line of the file, opening the file if necessary. `getservent()` opens and rewinds the file. If the *stayopen* flag is non-zero, the net data base will not be closed after each call to `getservent()` (either directly, or indirectly through one of the other "getserv" calls).

Successive calls to `getservent()` return either successive entries or `NULL`, indicating the end of the enumeration.

`endservent()` closes the file. `endservent()` may be called to indicate that the caller expects to do no further service entry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more service entry retrieval functions after calling `endservent()`.

Reentrant Interfaces

The functions `getservbyname()`, `getservbyport()`, and `getservent()` use static storage that is re-used in each call, making these functions unsafe for use in multithreaded applications.

The functions `getservbyname_r()`, `getservbyport_r()`, and `getservent_r()` provide reentrant interfaces for these operations.

Each reentrant interface performs the same operation as its non-reentrant counterpart, named by removing the "`_r`" suffix. The reentrant interfaces, however, use buffers supplied by the caller to store returned results, and are safe for use in both single-threaded and multithreaded applications.

Each reentrant interface takes the same parameters as its non-reentrant counterpart, as well as the following additional parameters. The parameter *result* must be a pointer to a `struct servent` structure allocated by the caller. On successful completion, the function returns the service entry in this structure. The parameter *buffer* must be a pointer to a buffer supplied by the

caller. This buffer is used as storage space for the service entry data. All of the pointers within the returned `struct servent` *result* point to data stored within this buffer. See the RETURN VALUES section of this man page. The buffer must be large enough to hold all of the data associated with the service entry. The parameter *buflen* should give the size in bytes of the buffer indicated by *buffer*.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. `setservent()` may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to `getservent_r()`, the threads will enumerate disjoint subsets of the service database.

Like their non-reentrant counterparts, `getservbyname_r()` and `getservbyport_r()` leave the enumeration position in an indeterminate state.

RETURN VALUES

Service entries are represented by the `struct servent` structure defined in `<netdb.h>`:

```
struct servent {
    char*s_name; /* official name of service */
    char*s_aliases; /* alias list */
    int*s_port; /* port service resides at */
    char*s_proto; /* protocol to use */
};
```

The members of this structure are:

| | |
|------------------------|--|
| <code>s_name</code> | The official name of the service. |
| <code>s_aliases</code> | A zero terminated list of alternate names for the service. \0 |
| <code>s_port</code> | The port number at which the service resides. Port numbers are \0 returned in network byte order. \0 |
| <code>s_proto</code> | The name of the protocol to use when contacting the service |

The functions `getservbyname()`, `getservbyname_r()`, `getservbyport()`, and `getservbyport_r()` each return a pointer to a `struct servent` if they successfully locate the requested entry; otherwise they return `NULL`.

The functions `setservent()` and `getservent_r()` each return a pointer to a `struct servent` if they successfully enumerate an entry; otherwise they return `NULL`, indicating the end of the enumeration.

The functions **getservbyname()** , **getservbyport()** , and **getservent()** use static storage, so returned data must be copied before a subsequent call to any of these functions if the data is to be saved.

When the pointer returned by the reentrant functions **getservbyname_r()** , **getservbyport_r()** , and **getservent_r()** is non-null, it is always equal to the *result* pointer that was supplied by the caller.

ERRORS

The reentrant functions **getservbyname_r()** , **getservbyport_r()** and **getservent_r()** will return `NULL` and set `errno` to `ERANGE` if the length of the buffer supplied by caller is not large enough to store the result. See **intro(2)** for the proper usage and interpretation of `errno` in multithreaded applications.

FILES

- `/etc/services` Internet network services
- `/etc/netconfig` network configuration file
- `/etc/nsswitch.conf` configuration file for the name-service switch

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|--|
| MT-Level | See "Reentrant Interfaces" in DESCRIPTION . |

SEE ALSO

intro(2) , **intro(3)** , **byteorder(3N)** , **netdir(3N)** , **netconfig(4)** , **nsswitch.conf(4)** , **services(4)** , **attributes(5)** , **netdb(5)**

WARNINGS

The reentrant interfaces **getservbyname_r()** , **getservbyport_r()** , and **getservent_r()** are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.

NOTES

The functions that return `struct servent` return the least significant 16-bits of the *s_port* field in *network byte order* . **getservbyport()** and **getservbyport_r()** also expect the input parameter *port* in the *network byte order* . See **htons(3N)** for more details on converting between host and network byte orders.

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

In order to ensure that they all return consistent results, **getservbyname()** , **getservbyname_r()** , and **netdir_getbyname()** are implemented in terms of the same internal library function. This function obtains the system-wide source

lookup policy based on the `inet` family entries in `netconfig(4)` and the `services` entry in `nsswitch.conf(4)`. Similarly, `getservbyport()`, `getservbyport_r()`, and `netdir_getbyaddr()` are implemented in terms of the same internal library function. If the `inet` family entries in `netconfig(4)` have a “-” in the last column for `nametoaddr` libraries, then the entry for `services` in `nsswitch.conf` will be used; otherwise the `nametoaddr` libraries in that column will be used, and `nsswitch.conf` will not be consulted.

There is no analogue of `getservent()` and `getservent_r()` in the `netdir` functions, so these enumeration functions go straight to the `services` entry in `nsswitch.conf`. Thus enumeration may return results from a different source than that used by `getservbyname()`, `getservbyname_r()`, `getservbyport()`, and `getservbyport_r()`.

When compiling multithreaded applications, see `intro(3)`, *Notes On Multithread Applications*, for information about the use of the `_REENTRANT` flag.

Use of the enumeration interfaces `getservent()` and `getservent_r()` is discouraged; enumeration may not be supported for all database sources. The semantics of enumeration are discussed further in `nsswitch.conf(4)`.

| NAME | getsockname – get socket name | | | | |
|----------------------|--|----------------|-----------------|----------|------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lsocket -lnsl [library ...] #include <sys/types.h> #include <sys/socket.h></pre> <pre>int getsockname(int s, struct sockaddr *name, socklen_t *namelen);</pre> | | | | |
| DESCRIPTION | getsockname() returns the current <i>name</i> for socket <i>s</i> . The <i>namelen</i> parameter should be initialized to indicate the amount of space pointed to by <i>name</i> . On return it contains the actual size in bytes of the name returned. | | | | |
| RETURN VALUES | If successful, getsockname() returns 0; otherwise it returns -1 and sets <i>errno</i> to indicate the error. | | | | |
| ERRORS | The call succeeds unless: <ul style="list-style-type: none"> EBADF The argument <i>s</i> is not a valid file descriptor. ENOMEM There was insufficient memory available for the operation to complete. ENOSR There were insufficient STREAMS resources available for the operation to complete. ENOTSOCK The argument <i>s</i> is not a socket. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe | | | | |
| SEE ALSO | bind(3N) , getpeername(3N) , socket(3N) , attributes(5) | | | | |

| NAME | getsockname – get the socket name | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lxnet [<i>library ...</i>] #include <sys/socket.h></pre> <pre>int getsockname(int <i>socket</i>, struct sockaddr *<i>address</i>, socklen_t *<i>address_len</i>);</pre> | | | | |
| DESCRIPTION | <p>The getsockname() function retrieves the locally-bound name of the specified socket, stores this address in the <code>sockaddr</code> structure pointed to by the <i>address</i> argument, and stores the length of this address in the object pointed to by the <i>address_len</i> argument.</p> <p>If the actual length of the address is greater than the length of the supplied <code>sockaddr</code> structure, the stored address will be truncated.</p> <p>If the socket has not been bound to a local name, the value stored in the object pointed to by <i>address</i> is unspecified.</p> | | | | |
| RETURN VALUES | Upon successful completion, 0 is returned, the <i>address</i> argument points to the address of the socket, and the <i>address_len</i> argument points to the length of the address. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error. | | | | |
| ERRORS | <p>The getsockname() function will fail:</p> <p>EBADF The <i>socket</i> argument is not a valid file descriptor.</p> <p>EFAULT The <i>address</i> or <i>address_len</i> parameter can not be accessed or written.</p> <p>ENOTSOCK The <i>socket</i> argument does not refer to a socket.</p> <p>EOPNOTSUPP The operation is not supported for this socket's protocol.</p> <p>The getsockname() function may fail if:</p> <p>EINVAL The socket has been shut down.</p> <p>ENOBUFS Insufficient resources were available in the system to complete the call.</p> <p>ENOSR There were insufficient STREAMS resources available for the operation to complete.</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |

getsockname(3XN)

X/Open Networking Services Library Functions

SEE ALSO

accept(3XN), bind(3XN), getpeername(3XN), socket(3XN)
attributes(5)

| | |
|--------------------|---|
| NAME | getsockopt, setsockopt – get and set options on sockets |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lsocket -lnsl [<i>library</i> ...] #include <sys/types.h> #include <sys/socket.h> int getsockopt(int s, int level, int optname, void * optval, socklen_t * optlen); int setsockopt(int s, int level, int optname, const void * optval, socklen_t optlen); </pre> |
| DESCRIPTION | <p>getsockopt() and setsockopt() manipulate options associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost “socket” level.</p> <p>When manipulating socket options, the level at which the option resides and the name of the option must be specified. To manipulate options at the “socket” level, <i>level</i> is specified as <code>SOL_SOCKET</code>. To manipulate options at any other level, <i>level</i> is the protocol number of the protocol that controls the option. For example, to indicate that an option is to be interpreted by the TCP protocol, <i>level</i> is set to the TCP protocol number (see getprotobyname(3N)).</p> <p>The parameters <i>optval</i> and <i>optlen</i> are used to access option values for setsockopt(). For getsockopt(), they identify a buffer in which the value(s) for the requested option(s) are to be returned. For setsockopt(), <i>optlen</i> is a value-result parameter, initially containing the size of the buffer pointed to by <i>optval</i>, and modified on return to indicate the actual size of the value returned. Use a 0 <i>optval</i> if no option value is to be supplied or returned.</p> <p><i>optname</i> and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The include file <code><sys/socket.h></code> contains definitions for the socket-level options described below. Options at other protocol levels vary in format and name.</p> <p>Most socket-level options take an int for <i>optval</i>. For setsockopt(), the <i>optval</i> parameter should be non-zero to enable a boolean option, or zero if the option is to be disabled. <code>SO_LINGER</code> uses a <code>struct linger</code> parameter that specifies</p> |

the desired state of the option and the linger interval (see below).
`struct linger` is defined in `<sys/socket.h>`. `struct linger` contains the following members:

`l_onoff` `on = 1/off = 0`

`l_linger` linger time, in seconds

The following options are recognized at the socket level. Except as noted, each may be examined with `getsockopt()` and set with `setsockopt()`.

`SO_DEBUG` enable/disable recording of debugging information

`SO_REUSEADDR` enable/disable local address reuse

`SO_KEEPAIVE` enable/disable keep connections alive

`SO_DONTROUTE` enable/disable routing bypass for outgoing messages

`SO_LINGER` linger on close if data is present

`SO_BROADCAST` enable/disable permission to transmit broadcast messages

`SO_OOBINLINE` enable/disable reception of out-of-band data in band

`SO_SNDBUF` set buffer size for output

`SO_RCVBUF` set buffer size for input

`SO_DGRAM_ERRIND` application wants delayed error

`SO_TYPE` get the type of the socket (get only)

`SO_ERROR` get and clear error on the socket (get only)

`SO_DEBUG` enables debugging in the underlying protocol modules.

`SO_REUSEADDR` indicates that the rules used in validating addresses supplied in a `bind(3N)` call should allow reuse of local addresses. `SO_KEEPAIVE` enables the periodic transmission of messages on a connected socket. If the connected party fails to respond to these messages, the connection is considered broken and processes using the socket are notified using a `SIGPIPE` signal. `SO_DONTROUTE` indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address.

`SO_LINGER` controls the action taken when unsent messages are queued on a socket and a `close(2)` is performed. If the socket promises reliable delivery of data and `SO_LINGER` is set, the system will block the process on the `close()` attempt until it is able to transmit the data or until it decides it is unable to deliver the information (a timeout period, termed the linger interval, is specified in the `setsockopt()` call when `SO_LINGER` is requested). If `SO_LINGER` is disabled and a `close()` is issued, the system will process the `close()` in a manner that allows the process to continue as quickly as possible.

The option `SO_BROADCAST` requests permission to send broadcast datagrams on the socket. With protocols that support out-of-band data, the `SO_OOBINLINE` option requests that out-of-band data be placed in the normal data input queue as received; it will then be accessible with `recv()` or `read()` calls without the `MSG_OOB` flag.

`SO_SNDBUF` and `SO_RCVBUF` are options that adjust the normal buffer sizes allocated for output and input buffers, respectively. The buffer size may be increased for high-volume connections or may be decreased to limit the possible backlog of incoming data. SunOS sets the maximum buffer size for both UDP and TCP to 256 Kbytes.

By default, delayed errors (such as ICMP port unreachable packets) are returned only for connected datagram sockets. `SO_DGRAM_ERRIND` makes it possible to receive errors for datagram sockets that are not connected. When this option is set, certain delayed errors received after completion of a `sendto()` or `sendmsg()` operation will cause a subsequent `sendto()` or `sendmsg()` operation using the same destination address (`to` parameter) to fail with the appropriate error. See `send(3N)` .

Finally, `SO_TYPE` and `SO_ERROR` are options used only with `getsockopt()` . `SO_TYPE` returns the type of the socket (for example, `SOCK_STREAM`). It is useful for servers that inherit sockets on startup. `SO_ERROR` returns any pending error on the socket and clears the error status. It may be used to check for asynchronous errors on connected datagram sockets or for other asynchronous errors.

RETURN VALUES

If successful, `getsockopt()` returns 0 ; otherwise, it returns -1 and sets `errno` to indicate the error.

ERRORS

The call succeeds unless:

| | |
|--------------------|--|
| EBADF | The argument <code>s</code> is not a valid file descriptor. |
| ENOMEM | There was insufficient memory available for the operation to complete. |
| ENOPROTOOPT | The option is unknown at the level indicated. |

ENOSR There were insufficient STREAMS resources available for the operation to complete.

ENOTSOCK The argument *s* is not a socket.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

close(2), **ioctl(2)**, **read(2)**, **bind(3N)**, **getprotobyname(3N)**, **recv(3N)**, **send(3N)**, **socket(3N)**, **attributes(5)**

| | | | | | | | | | |
|----------------------|---|-----------------|--|----------------------|--|---------------------|--|---------------------|--|
| NAME | getsockopt – get the socket options | | | | | | | | |
| SYNOPSIS | <pre>cc [flag ...] file ... -lxnet [library ...] #include <sys/socket.h></pre> <p>int getsockopt(int <i>socket</i>, int <i>level</i>, int <i>option_name</i>, void *<i>option_value</i>, socklen_t *<i>option_len</i>);</p> | | | | | | | | |
| DESCRIPTION | <p>The getsockopt() function retrieves the value for the option specified by the <i>option_name</i> argument for the socket specified by the <i>socket</i> argument. If the size of the option value is greater than <i>option_len</i>, the value stored in the object pointed to by the <i>option_value</i> argument will be silently truncated. Otherwise, the object pointed to by the <i>option_len</i> argument will be modified to indicate the actual length of the value.</p> <p>The <i>level</i> argument specifies the protocol level at which the option resides. To retrieve options at the socket level, specify the <i>level</i> argument as SOL_SOCKET. To retrieve options at other levels, supply the appropriate protocol number for the protocol controlling the option. For example, to indicate that an option will be interpreted by the TCP (Transport Control Protocol), set <i>level</i> to the protocol number of TCP, as defined in the <netinet/in.h> header, or as determined by using getprotobyname(3XN) function.</p> <p>The socket in use may require the process to have appropriate privileges to use the getsockopt() function.</p> <p>The <i>option_name</i> argument specifies a single option to be retrieved. It can be one of the following values defined in <sys/socket.h>:</p> <table border="0"> <tr> <td style="padding-right: 20px;">SO_DEBUG</td> <td>Reports whether debugging information is being recorded. This option stores an <code>int</code> value. This is a boolean option.</td> </tr> <tr> <td>SO_ACCEPTCONN</td> <td>Reports whether socket listening is enabled. This option stores an <code>int</code> value.</td> </tr> <tr> <td>SO_BROADCAST</td> <td>Reports whether transmission of broadcast messages is supported, if this is supported by the protocol. This option stores an <code>int</code> value. This is a boolean option.</td> </tr> <tr> <td>SO_REUSEADDR</td> <td>Reports whether the rules used in validating addresses supplied to bind(3XN) should allow reuse of local addresses, if this is supported by the protocol. This option stores an <code>int</code> value. This is a boolean option.</td> </tr> </table> | SO_DEBUG | Reports whether debugging information is being recorded. This option stores an <code>int</code> value. This is a boolean option. | SO_ACCEPTCONN | Reports whether socket listening is enabled. This option stores an <code>int</code> value. | SO_BROADCAST | Reports whether transmission of broadcast messages is supported, if this is supported by the protocol. This option stores an <code>int</code> value. This is a boolean option. | SO_REUSEADDR | Reports whether the rules used in validating addresses supplied to bind(3XN) should allow reuse of local addresses, if this is supported by the protocol. This option stores an <code>int</code> value. This is a boolean option. |
| SO_DEBUG | Reports whether debugging information is being recorded. This option stores an <code>int</code> value. This is a boolean option. | | | | | | | | |
| SO_ACCEPTCONN | Reports whether socket listening is enabled. This option stores an <code>int</code> value. | | | | | | | | |
| SO_BROADCAST | Reports whether transmission of broadcast messages is supported, if this is supported by the protocol. This option stores an <code>int</code> value. This is a boolean option. | | | | | | | | |
| SO_REUSEADDR | Reports whether the rules used in validating addresses supplied to bind(3XN) should allow reuse of local addresses, if this is supported by the protocol. This option stores an <code>int</code> value. This is a boolean option. | | | | | | | | |

| | |
|---------------------|---|
| SO_KEEPAIVE | <p>Reports whether connections are kept active with periodic transmission of messages, if this is supported by the protocol.</p> <p>If the connected socket fails to respond to these messages, the connection is broken and processes writing to that socket are notified with a SIGPIPE signal. This option stores an <code>int</code> value.</p> <p>This is a boolean option.</p> |
| SO_LINGER | <p>Reports whether the socket lingers on <code>close(2)</code> if data is present. If <code>SO_LINGER</code> is set, the system blocks the process during <code>close(2)</code> until it can transmit the data or until the end of the interval indicated by the <code>l_linger</code> member, whichever comes first. If <code>SO_LINGER</code> is not specified, and <code>close(2)</code> is issued, the system handles the call in a way that allows the process to continue as quickly as possible. This option stores a <code>linger</code> structure.</p> |
| SO_OOINLINE | <p>Reports whether the socket leaves received out-of-band data (data marked urgent) in line. This option stores an <code>int</code> value. This is a boolean option.</p> |
| SO_SNDBUF | <p>Reports send buffer size information. This option stores an <code>int</code> value.</p> |
| SO_RCVBUF | <p>Reports receive buffer size information. This option stores an <code>int</code> value.</p> |
| SO_ERROR | <p>Reports information about error status and clears it. This option stores an <code>int</code> value.</p> |
| SO_TYPE | <p>Reports the socket type. This option stores an <code>int</code> value.</p> |
| SO_DONTROUTE | <p>Reports whether outgoing messages bypass the standard routing facilities. The destination must be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. The effect, if any, of this option depends on what</p> |

protocol is in use. This option stores an `int` value. This is a boolean option.

For boolean options, a zero value indicates that the option is disabled and a non-zero value indicates that the option is enabled.

Options at other protocol levels vary in format and name.

The socket in use may require the process to have appropriate privileges to use the `getsockopt()` function.

RETURN VALUES

Upon successful completion, `getsockopt()` returns 0. Otherwise, `-1` is returned and `errno` is set to indicate the error.

ERRORS

The `getsockopt()` function will fail if:

- EBADF** The *socket* argument is not a valid file descriptor.
 - EFAULT** The *option_value* or *option_len* parameter can not be accessed or written.
 - EINVAL** The specified option is invalid at the specified socket level.
 - ENOPROTOOPT** The option is not supported by the protocol.
 - ENOTSOCK** The *socket* argument does not refer to a socket.
- The `getsockopt()` function may fail if:
- EACCES** The calling process does not have the appropriate privileges.
 - EINVAL** The socket has been shut down.
 - ENOBUFS** Insufficient resources are available in the system to complete the call.
 - ENOSR** There were insufficient STREAMS resources available for the operation to complete.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`close(2)`, `bind(3XN)`, `endprotoent(3XN)`, `setsockopt(3XN)`, `socket(3XN)`, `attributes`

| | |
|--------------------|--|
| NAME | getspnam, getspnam_r, getspent, getspent_r, setspent, endspent, fgetspent, fgetspent_r – get password entry |
| SYNOPSIS | <pre>#include <shadow.h> struct spwd * getspnam(const char * <i>name</i>); struct spwd * getspnam_r(const char * <i>name</i>, struct spwd * <i>result</i>, char * <i>buffer</i>, int <i>buflen</i>); struct spwd * getspent(void); struct spwd * getspent_r(struct spwd * <i>result</i>, char * <i>buffer</i>, int <i>buflen</i>); void setspent(void); void endspent(void); struct spwd * fgetspent(FILE * <i>fp</i>); struct spwd * fgetspent_r(FILE * <i>fp</i>, struct spwd * <i>result</i>, char * <i>buffer</i>, int <i>buflen</i>);</pre> |
| DESCRIPTION | <p>These functions are used to obtain shadow password entries. An entry may come from any of the sources for <code>shadow</code> specified in the <code>/etc/nsswitch.conf</code> file (see <code>nsswitch.conf(4)</code>).</p> <p>The <code>getspnam()</code> function searches for a shadow password entry with the login name specified by the character string argument <code>name</code>.</p> <p>The <code>setspent()</code>, <code>getspent()</code>, and <code>endspent()</code> functions are used to enumerate shadow password entries from the database.</p> <p>The <code>setspent()</code> function sets (or resets) the enumeration to the beginning of the set of shadow password entries. This function should be called before the first call to <code>getspent()</code>. Calls to <code>getspnam()</code> leave the enumeration position in an indeterminate state.</p> <p>Successive calls to <code>getspent()</code> return either successive entries or <code>NULL</code>, indicating the end of the enumeration.</p> <p>The <code>endspent()</code> function may be called to indicate that the caller expects to do no further shadow password retrieval operations; the system may then close the shadow password file, deallocate resources it was using, and so forth. It is still allowed, but possibly less efficient, for the process to call more shadow password functions after calling <code>endspent()</code>.</p> <p>The <code>fgetspent()</code> function, unlike the other functions above, does not use <code>nsswitch.conf</code>; it reads and parses the next line from the stream <code>fp</code>, which is assumed to have the format of the <code>shadow</code> file (see <code>shadow(4)</code>).</p> |

Reentrant Interfaces

The **getspnam()** , **getspent()** , and **fgetspent()** functions use static storage that is re-used in each call, making these routines unsafe for use in multithreaded applications.

The **getspnam_r()** , **getspent_r()** , and **fgetspent_r()** functions provide reentrant interfaces for these operations.

Each reentrant interface performs the same operation as its non-reentrant counterpart, named by removing the `_r` suffix. The reentrant interfaces, however, use buffers supplied by the caller to store returned results, and are safe for use in both single-threaded and multithreaded applications.

Each reentrant interface takes the same argument as its non-reentrant counterpart, as well as the following additional arguments. The *result* argument must be a pointer to a `struct spwd` structure allocated by the caller. On successful completion, the function returns the shadow password entry in this structure. The *buffer* argument must be a pointer to a buffer supplied by the caller. This buffer is used as storage space for the shadow password data. All of the pointers within the returned `struct spwd` *result* point to data stored within this buffer (see RETURN VALUES). The buffer must be large enough to hold all of the data associated with the shadow password entry. The *buflen* argument should give the size in bytes of the buffer indicated by *buffer*.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. The **setspent()** function may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to **getspent_r()** , the threads will enumerate disjoint subsets of the shadow password database.

Like its non-reentrant counterpart, **getspnam_r()** leaves the enumeration position in an indeterminate state.

RETURN VALUES

Password entries are represented by the `struct spwd` structure defined in `<shadow.h>` :

```
struct spwd{
    char          *sp_namp;      /* login name */
    char          *sp_pwdp;     /* encrypted passwd */
    long          sp_lstchg;     /* date of last change */
    long          sp_min;       /* min days to passwd change */
    long          sp_max;       /* max days to passwd change*/
    long          sp_warn;      /* warning period */
    long          sp_inact;     /* max days inactive */
    long          sp_expire;    /* account expiry date */
    unsigned long sp_flag;     /* not used */
};
```

See **shadow(4)** for more information on the interpretation of this data.

The **getspnam()** and **getspnam_r()** functions each return a pointer to a `struct spwd` if they successfully locate the requested entry; otherwise they return `NULL`.

The **getspent()**, **getspent_r()**, **fgetspent()**, and **fgetspent_r()** functions each return a pointer to a `struct spwd` if they successfully enumerate an entry; otherwise they return `NULL`, indicating the end of the enumeration.

The **getspnam()**, **getspent()**, and **fgetspent()** functions use static storage, so returned data must be copied before a subsequent call to any of these functions if the data is to be saved.

When the pointer returned by the reentrant functions **getspnam_r()**, **getspent_r()**, and **fgetspent_r()** is non-null, it is always equal to the *result* pointer that was supplied by the caller.

ERRORS

The reentrant functions **getspnam_r()**, **getspent_r()**, and **fgetspent_r()** will return `NULL` and set `errno` to `ERANGE` if the length of the buffer supplied by caller is not large enough to store the result. See **intro(2)** for the proper usage and interpretation of `errno` in multithreaded applications.

USAGE

Applications that use the interfaces described on this manual page cannot be linked statically, since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|--|
| MT-Level | See "Reentrant Interfaces" in DESCRIPTION . |

SEE ALSO

nispasswd(1), **passwd(1)**, **yppasswd(1)**, **intro(3)**, **getlogin(3C)**, **getpwnam(3C)**, **nsswitch.conf(4)**, **passwd(4)**, **shadow(4)**, **attributes(5)**

WARNINGS

The reentrant interfaces **getspnam_r()**, **getspent_r()**, and **fgetspent_r()** are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.

NOTES

When compiling multithreaded applications, see **intro(3)**, *Notes On Multithreaded Applications*, for information about the use of the `_REENTRANT` flag.

Use of the enumeration interfaces **getspent()** and **getspent_r()** is not recommended; enumeration is supported for the shadow file, NIS, and NIS+, but in general is not efficient and may not be supported for all database sources. The semantics of enumeration are discussed further in **nsswitch.conf(4)**.

Access to shadow password information may be restricted in a manner depending on the database source being used. Access to the `/etc/shadow` file is generally restricted to processes running as the super-user (root). Other database sources may impose stronger or less stringent restrictions.

When NIS is used as the database source, the information for the shadow password entries is obtained from the “passwd.byname” map. This map stores only the information for the `sp_namp` and `sp_pwdp` fields of the `struct spwd` structure. Shadow password entries obtained from NIS will contain the value `-1` in the remainder of the fields.

When NIS+ is used as the database source, and the caller lacks the permission needed to retrieve the encrypted password from the NIS+ “passwd.org_dir” table, the NIS+ service returns the string “*NP*” instead of the actual encrypted password string. The functions described on this page will then return the string “*NP*” to the caller as the value of the member `sp_pwdp` in the returned shadow password structure.

| | |
|----------------------|--|
| NAME | getsubopt – parse suboptions from a string |
| SYNOPSIS | <pre>#include <stdlib.h> int getsubopt(char **optionp, char * const *tokens, char **valuep);</pre> |
| DESCRIPTION | <p>The getsubopt() function parses suboptions in a flag argument that was initially parsed by getopt(3C). The suboptions are separated by commas and may consist of either a single token or a token-value pair separated by an equal sign. Since commas delimit suboptions in the option string, they are not allowed to be part of the suboption or the value of a suboption; if present in the option input string, they are changed to null characters. White spaces within tokens or token-value pairs must be protected from the shell by quotes.</p> <p>The syntax described above is used in the following example by the mount(1M), utility, which allows the user to specify mount parameters with the -o option as follows:</p> <pre>mount -o rw,hard,bg,wsiz=1024 speed:/usr /usr</pre> <p>In this example there are four suboptions: rw, hard, bg, and wsiz, the last of which has an associated value of 1024.</p> <p>The getsubopt() function takes the address of a pointer to the option string, a vector of possible tokens, and the address of a value string pointer. It returns the index of the token that matched the suboption in the input string, or -1 if there was no match. If the option string pointed to by <i>optionp</i> contains only one suboption, getsubopt() updates <i>optionp</i> to point to the null character at the end of the string; otherwise it isolates the suboption by replacing the comma separator with a null character, and updates <i>optionp</i> to point to the start of the next suboption. If the suboption has an associated value, getsubopt() updates <i>valuep</i> to point to the value's first character. Otherwise it sets <i>valuep</i> to NULL.</p> <p>The token vector is organized as a series of pointers to null strings. The end of the token vector is identified by a null pointer.</p> <p>When getsubopt() returns, a non-null value for <i>valuep</i> indicates that the suboption that was processed included a value. The calling program may use this information to determine if the presence or absence of a value for this suboption is an error.</p> <p>When getsubopt() fails to match the suboption with the tokens in the <i>tokens</i> array, the calling program should decide if this is an error, or if the unrecognized option should be passed to another program.</p> |
| RETURN VALUES | The getsubopt() function returns -1 when the token it is scanning is not in the token vector. The variable addressed by <i>valuep</i> contains a pointer to the first |

character of the token that was not recognized, rather than a pointer to a value for that token.

The variable addressed by *optionp* points to the next option to be parsed, or a null character if there are no more options.

EXAMPLES

EXAMPLE 1 Example of `getsubopt()` function.

The following example demonstrates the processing of options to the `mount(1M)` utility using `getsubopt()`.

```
#include <stdlib.h>

char *myopts[] = {
#define READONLY      0
    "ro",
#define READWRITE     1
    "rw",
#define WRITESIZE     2
    "wsize",
#define READSIZE      3
    "rsize",
    NULL};

main(argc, argv)
int  argc;
char **argv;
{
    int sc, c, errflag;
    char *options, *value;
    extern char *optarg;
    extern int optind;
    .
    .
    while((c = getopt(argc, argv, "abf:o:")) != -1) {
        switch (c) {
            case 'a': /* process a option */
                break;
            case 'b': /* process b option */
                break;
            case 'f':
                ofile = optarg;
                break;
            case '?':
                errflag++;
                break;
            case 'o':
                options = optarg;
                while (*options != '\0') {
                    switch(getsubopt(&options, myopts, &value)){
                        case READONLY : /* process ro option */
                            break;
                        case READWRITE : /* process rw option */
                            break;
                    }
                }
        }
    }
}
```

```

                                case WRITESIZE : /* process wsize option */
                                if (value == NULL) {
                                    error_no_arg();
                                    errflag++;
                                } else
                                    write_size = atoi(value);
                                break;
                                case READSIZE : /* process rsize option */
                                if (value == NULL) {
                                    error_no_arg();
                                    errflag++;
                                } else
                                    read_size = atoi(value);
                                break;
                                default :
                                    /* process unknown token */
                                    error_bad_token(value);
                                    errflag++;
                                    break;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    if (errflag) {
        /* print usage instructions etc. */
    }
    for (; optind < argc; optind++) {
        /* process remaining arguments */
    }
    .
    .
    .
}

```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

mount(1M), **getopt(3C)**, **attributes(5)**

| | |
|--------------------|---|
| NAME | gettext, dgettext, dcgettext, textdomain, bindtextdomain – message handling functions |
| SYNOPSIS | <pre>#include <libintl.h> char * gettext(const char * msgid); char * dgettext(const char * domainname, const char * msgid); char * textdomain(const char * domainname); char * bindtextdomain(const char * domainname, const char * dirname); #include <libintl.h> #include <locale.h> char * dcgettext(const char * domainname, const char * msgid, int category);</pre> |
| DESCRIPTION | <p>The gettext() , dgettext() , and dcgettext() functions attempt to retrieve a target string based on the specified <i>msgid</i> argument within the context of a specific domain and the current locale. The length of strings returned by gettext() , dgettext() , and dcgettext() is undetermined until the function is called. The <i>msgid</i> argument is a null-terminated string.</p> <p>The NLSPATH environment variable (see environ(5)) is searched first for the location of the LC_MESSAGES catalogue. The setting of the LC_MESSAGES category of the current locale determines the locale used by gettext() and dgettext() for string retrieval. The <i>category</i> argument determines the locale used by dcgettext() . If NLSPATH is not defined and the current locale is "C", gettext() , dgettext() , and dcgettext() simply return the message string that was passed. In a locale other than "C", if NLSPATH is not defined or if a message catalogue is not found in any of the components specified by NLSPATH , the routines search for the message catalogue <i>dirname / locale / category / domainname .mo</i> , after querying bindtextdomain() for <i>dirname</i> .</p> <p>For gettext() , the domain used is set by the last valid call to textdomain() . If a valid call to textdomain() has not been made, the default domain (called <i>messages</i>) is used.</p> <p>For dgettext() and dcgettext() , the domain used is specified by the <i>domainname</i> argument. The <i>domainname</i> argument is equivalent in syntax and meaning to the <i>domainname</i> argument to textdomain() , except that the selection of the domain is valid only for the duration of the dgettext() or dcgettext() function call.</p> <p>The textdomain() function sets or queries the name of the current domain of the active LC_MESSAGES locale category. The <i>domainname</i> argument is a</p> |

null-terminated string that can contain only the characters allowed in legal filenames.

The *domainname* argument is the unique name of a domain on the system. If there are multiple versions of the same domain on one system, namespace collisions can be avoided by using **bindtextdomain()**. If **textdomain()** is not called, a default domain is selected. The setting of domain made by the last valid call to **textdomain()** remains valid across subsequent calls to **setlocale(3C)**, and **gettext()**.

The *domainname* argument is applied to the currently active LC_MESSAGES locale.

The current setting of the domain can be queried without affecting the current state of the domain by calling **textdomain()** with *domainname* set to the null pointer. Calling **textdomain()** with a *domainname* argument of a null string sets the domain to the default domain (`messages`).

The **bindtextdomain()** function binds the path predicate for a message domain *domainname* to the value contained in *dirname*. If *domainname* is a non-empty string and has not been bound previously, **bindtextdomain()** binds *domainname* with *dirname*.

If *domainname* is a non-empty string and has been bound previously, **bindtextdomain()** replaces the old binding with *dirname*. The *dirname* argument can be an absolute or relative pathname being resolved when **gettext()**, **dgettext()**, or **dcgettext()** are called. If *domainname* is a null pointer or an empty string, **bindtextdomain()** returns `NULL`. User defined domain names cannot begin with the string `SYS_`. Domain names beginning with this string are reserved for system use.

RETURN VALUES

The individual bytes of the string returned by **gettext()**, **dgettext()**, or **dcgettext()** can contain any value other than null. If *msgid* is a null pointer, the return value is undefined. The string returned must not be modified by the program, and can be invalidated by a subsequent call to **gettext()**, **dgettext()**, **dcgettext()**, or **setlocale(3C)**. If the *domainname* argument to **dgettext()** or **dcgettext()** is a null pointer, the results are undefined.

If the target string cannot be found in the current locale and selected domain, **gettext()**, **dgettext()**, and **dcgettext()** return *msgid*.

The normal return value from **textdomain()** is a pointer to a string containing the current setting of the domain. If *domainname* is a null pointer, **textdomain()** returns a pointer to the string containing the current domain. If **textdomain()** was not previously called and *domainname* is a null string, the name of the default domain is returned. The name of the default domain is `messages`.

The return value from **bindtextdomain()** is a null-terminated string containing *dirname* or the directory binding associated with *domainname* if *dirname* is `NULL`. If no binding is found, the default return value is `/usr/lib/locale`. If *domainname* is a null pointer or an empty string, **bindtextdomain()** takes no action and returns a null pointer. The string returned must not be modified by the caller.

USAGE

These routines impose no limit on message length. However, a text *domainname* is limited to `TEXTDOMAINMAX` (256) bytes.

The **gettext()**, **dgettext()**, **dcgettext()**, **textdomain()**, and **bindtextdomain()** can be used safely in multithreaded applications, as long as **setlocale(3C)** is not being called to change the locale.

FILES

`/usr/lib/locale`

The default path predicate for message domain files.

`/usr/lib/locale/ locale /LC_MESSAGES/ domainname .mo`

system default location for file containing messages for language *locale* and *domainname*

`/usr/lib/locale/ locale /LC_XXX/ domainname .mo`

system default location for file containing messages for language *locale* and *domainname* for **dcgettext()** calls where `LC_XXX` is `LC_CTYPE`, `LC_NUMERIC`, `LC_TIME`, `LC_COLLATE`, `LC_MONETARY`, or `LC_MESSAGES`.

`dirname / locale /LC_MESSAGES/ domainname .mo`

location for file containing messages for domain *domainname* and path predicate *dirname* after a successful call to **bindtextdomain()**

`dirname / locale /LC_XXX/ domainname .mo`

location for files containing messages for domain *domainname*, language *locale*, and path predicate *dirname* after a successful call to **bindtextdomain()** for **dcgettext()** calls where `LC_XXX` is one of `LC_CTYPE`, `LC_NUMERIC`, `LC_TIME`, `LC_COLLATE`, `LC_MONETARY`, or `LC_MESSAGES`.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|----------------------|
| MT-Level | Safe with exceptions |

SEE ALSO

`msgfmt(1)`, `xgettext(1)`, `setlocale(3C)`, `attributes(5)`, `environ(5)`

| | |
|--------------------|---|
| NAME | gettimeofday, settimeofday – get or set the date and time |
| SYNOPSIS | <pre> /usr/ucb/cc [flag ...] file ... #include <sys/time.h> int gettimeofday(tp, tzp); struct timeval * tzp ; struct timezone * tzp ; int settimeofday(tp, tzp); struct timeval * tzp ; struct timezone * tzp ; </pre> |
| DESCRIPTION | <p>The system's notion of the current Greenwich time is obtained with the gettimeofday() call, and set with the settimeofday() call. The current time is expressed in elapsed seconds and microseconds since 00:00 GMT, January 1, 1970 (zero hour). The resolution of the system clock is hardware dependent; the time may be updated continuously, or in clock ticks.</p> <pre> long\011tv_sec;\011 /* seconds since Jan. 1, 1970 */ long\011tv_usec; \011/* and microseconds */ </pre> <p><i>tp</i> points to a <code>timeval</code> structure, which includes the following members:</p> <p>If <i>tp</i> is a NULL pointer, the current time information is not returned or set.</p> <p><i>tzp</i> is an obsolete pointer formerly used to get and set timezone information. <i>tzp</i> is now ignored. Timezone information is now handled using the TZ environment variable; see TIMEZONE(4) .</p> |

Only the privileged user may set the time of day.

RETURN VALUES

A `-1` return value indicates an error occurred; in this case an error code is stored in the global variable `errno`.

ERRORS

The following error codes may be set in `errno`:

EINVAL *tp* specifies an invalid time.

EPERM A user other than the privileged user attempted to set the time.

SEE ALSO

`adjtime(2)`, `ctime(3C)`, `gettimeofday(3C)`, `TIMEZONE(4)`

NOTES

Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

tzp is ignored in SunOS 5. X releases.

tv_usec is always 0.

| | |
|----------------------|---|
| NAME | gettimeofday, settimeofday – get or set the date and time |
| SYNOPSIS | <pre>#include <sys/time.h> int gettimeofday(struct timeval * tp, void *); int settimeofday(struct timeval * tp, void *);</pre> |
| DESCRIPTION | <p>The gettimeofday() function gets and the settimeofday() function sets the system's notion of the current time. The current time is expressed in elapsed seconds and microseconds since 00:00 Universal Coordinated Time, January 1, 1970. The resolution of the system clock is hardware dependent; the time may be updated continuously or in clock ticks.</p> <p>The <i>tp</i> argument points to a <code>timeval</code> structure, which includes the following members:</p> <pre>long tv_sec; /* seconds since Jan. 1, 1970 */ long tv_usec; /* and microseconds */</pre> <p>If <i>tp</i> is a null pointer, the current time information is not returned or set.</p> <p>The TZ environment variable holds time zone information. See TIMEZONE(4).</p> <p>The second argument to gettimeofday() and settimeofday() should be a pointer to <code>NULL</code>.</p> <p>Only the super-user may set the time of day.</p> |
| RETURN VALUES | Upon successful completion, 0 is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error. |
| ERRORS | <p>The gettimeofday() function will fail if:</p> <p>EINVAL The structure pointed to by <i>tp</i> specifies an invalid time.</p> <p>EPERM A user other than the privileged user attempted to set the time or time zone.</p> <p>Additionally, the gettimeofday() function will fail for 32-bit interfaces if:</p> <p>E_OVERFLOW The system time has progressed beyond 2038, thus the size of the <code>tv_sec</code> member of the <code>timeval</code> structure pointed to by <i>tp</i> is insufficient to hold the current time in seconds.</p> |
| USAGE | If the <code>tv_usec</code> member of <i>tp</i> is > 500000, settimeofday() rounds the seconds upward. If the time needs to be set with better than one second accuracy, call settimeofday() for the seconds and then adjtime(2) for finer accuracy. |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

adjtime(2), **ctime(3C)**, **TIMEZONE(4)**, **attributes(5)**

| | |
|----------------------|---|
| NAME | gettext – retrieve a text string |
| SYNOPSIS | <pre>#include <nl_types.h> char *gettext(const char *msgid, const char *dflt_str);</pre> |
| DESCRIPTION | <p>The gettext() function retrieves a text string from a message file. The arguments to the function are a message identification <i>msgid</i> and a default string <i>dflt_str</i> to be used if the retrieval fails.</p> <p>The text strings are in files created by the mkmsgs utility (see mkmsgs(1)) and installed in directories in <code>/usr/lib/locale/<i>locale</i>/LC_MESSAGES</code>.</p> <p>The directory <code>locale</code> can be viewed as the language in which the text strings are written. The user can request that messages be displayed in a specific language by setting the environment variable <code>LC_MESSAGES</code>. If <code>LC_MESSAGES</code> is not set, the environment variable <code>LANG</code> will be used. If <code>LANG</code> is not set, the files containing the strings are in <code>/usr/lib/locale/C/LC_MESSAGES/*</code>.</p> <p>The user can also change the language in which the messages are displayed by invoking the setlocale(3C) function with the appropriate arguments.</p> <p>If gettext() fails to retrieve a message in a specific language it will try to retrieve the same message in U.S. English. On failure, the processing depends on what the second argument <i>dflt_str</i> points to. A pointer to the second argument is returned if the second argument is not the null string. If <i>dflt_str</i> points to the null string, a pointer to the U.S. English text string "Message not found!!\n" is returned.</p> <p>The following depicts the acceptable syntax of <i>msgid</i> for a call to gettext().</p> <pre><msgid> = <msgfilename>: <msgnumber></pre> <p>The first field is used to indicate the file that contains the text strings and must be limited to 14 characters. These characters must be selected from the set of all character values excluding <code>\0</code> (null) and the ASCII code for <code>/</code> (slash) and <code>:</code> (colon). The names of message files must be the same as the names of files created by mkmsgs and installed in <code>/usr/lib/locale/<i>locale</i>/LC_MESSAGES/*</code>. The numeric field indicates the sequence number of the string in the file. The strings are numbered from 1 to <i>n</i> where <i>n</i> is the number of strings in the file.</p> |
| RETURN VALUES | Upon failure to pass either the correct <i>msgid</i> or a valid message number to gettext() , a pointer to the text string "Message not found!!\n" is returned. |
| USAGE | It is recommended that gettext(3C) be used in place of this function. |

EXAMPLES

EXAMPLE 1 Example of **gettext()** function.

In the following example,

```
gettext("UX:10", "hello world\n")
gettext("UX:10", "")
```

UX is the name of the file that contains the messages and 10 is the message number.

FILES

/usr/lib/locale/C/LC_MESSAGES/*

contains default message files created by **mkmsgs**

/usr/lib/locale/*locale*/LC_MESSAGES/*

contains message files for different languages created by **mkmsgs**

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|----------------------|
| MT-Level | Safe with exceptions |

SEE ALSO

exstr(1), **mkmsgs(1)**, **srchtxt(1)**, **gettext(3C)**, **fmtmsg(3C)**, **setlocale(3C)**, **attributes(5)**, **environ(5)**

| | | | | | | | | | | | |
|---------------------------|--|--------------------------|---------------------------|---------------------------|---------------------------|----------------------|-----------------------|-----------------------|-----------------------|-----------------------|------------------------|
| NAME | getusershell, setusershell, endusershell – get legal user shells | | | | | | | | | | |
| SYNOPSIS | <pre>char * getusershell() void setusershell() void endusershell()</pre> | | | | | | | | | | |
| DESCRIPTION | <p>The getusershell() function returns a pointer to a legal user shell as defined by the system manager in the file <code>/etc/shells</code>. If <code>/etc/shells</code> does not exist, the following locations of the standard system shells are used in its place:</p> <table border="1" style="margin-left: 20px; border-collapse: collapse;"> <tr> <td style="padding: 2px;"><code>/usr/bin/sh</code></td> <td style="padding: 2px;"><code>/usr/bin/csh</code></td> </tr> <tr> <td style="padding: 2px;"><code>/usr/bin/ksh</code></td> <td style="padding: 2px;"><code>/usr/bin/jsh</code></td> </tr> <tr> <td style="padding: 2px;"><code>/bin/sh</code></td> <td style="padding: 2px;"><code>/bin/csh</code></td> </tr> <tr> <td style="padding: 2px;"><code>/bin/ksh</code></td> <td style="padding: 2px;"><code>/bin/jsh</code></td> </tr> <tr> <td style="padding: 2px;"><code>/sbin/sh</code></td> <td style="padding: 2px;"><code>/sbin/jsh</code></td> </tr> </table> <p>The getusershell() function opens the file <code>/etc/shells</code>, if it exists, and returns the next entry in the list of shells.</p> <p>The setusershell() function rewinds the file or the list.</p> <p>The endusershell() function closes the file, frees any memory used by getusershell() and setusershell(), and rewinds the file <code>/etc/shells</code>.</p> | <code>/usr/bin/sh</code> | <code>/usr/bin/csh</code> | <code>/usr/bin/ksh</code> | <code>/usr/bin/jsh</code> | <code>/bin/sh</code> | <code>/bin/csh</code> | <code>/bin/ksh</code> | <code>/bin/jsh</code> | <code>/sbin/sh</code> | <code>/sbin/jsh</code> |
| <code>/usr/bin/sh</code> | <code>/usr/bin/csh</code> | | | | | | | | | | |
| <code>/usr/bin/ksh</code> | <code>/usr/bin/jsh</code> | | | | | | | | | | |
| <code>/bin/sh</code> | <code>/bin/csh</code> | | | | | | | | | | |
| <code>/bin/ksh</code> | <code>/bin/jsh</code> | | | | | | | | | | |
| <code>/sbin/sh</code> | <code>/sbin/jsh</code> | | | | | | | | | | |
| RETURN VALUES | The getusershell() function returns a null pointer on EOF. | | | | | | | | | | |
| BUGS | All information is contained in memory that may be freed with a call to endusershell() , so it must be copied if it is to be saved. | | | | | | | | | | |

| | |
|--------------------|---|
| NAME | getutent, getutid, getutline, pututline, setutent, endutent, utmpname – access utmp file entry |
| SYNOPSIS | <pre>#include <utmp.h> struct utmp * getutent(void); struct utmp * getutid(const struct utmp * id); struct utmp * getutline(const struct utmp * line); struct utmp * pututline(const struct utmp * utmp); void setutent(void); void endutent(void); int utmpname(const char * file);</pre> |
| DESCRIPTION | <p>The getutent() , getutid() , getutline() , and pututline() functions each return a pointer to a utmp structure with the following members:</p> <pre>char ut_user[8]; /* user login name */ char ut_id[4]; /* /sbin/inittab id (usually line #) */ char ut_line[12]; /* device name (console, lnxx) */ short ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ time_t ut_time; /* time entry was made */</pre> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short e_termination; /* termination status */ short e_exit; /* exit status */</pre> <p>getutent() The getutent() function reads in the next entry from a utmp -like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p>getutid() The getutid() function searches forward from the current point in the utmp file until it finds an entry with a <code>ut_type</code> matching <code>id ⇒ ut_type</code> if the type specified is <code>RUN_LVL</code> , <code>BOOT_TIME</code> , <code>OLD_TIME</code> , or <code>NEW_TIME</code> . If the type specified in <code>id</code> is <code>INIT_PROCESS</code> , <code>LOGIN_PROCESS</code> , <code>USER_PROCESS</code> , or <code>DEAD_PROCESS</code> , then getutid() will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <code>id ⇒ ut_id</code>. If the end of file is reached without a match, it fails.</p> <p>getutline() The getutline() function searches forward from the current point in the utmp file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>ut_line</code> string</p> |

matching the `line` \Rightarrow `ut_line` string. If the end of file is reached without a match, it fails.

pututline() The **pututline()** function writes the supplied `utmp` structure into the `utmp` file. It uses **getutid()** to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of **pututline()** will have searched for the proper entry using one of the these functions. If so, **pututline()** will not search. If **pututline()** does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the `utmp` structure. When called by a non-root user, **pututline()** invokes a **setuid()** root program to verify and write the entry, since `/etc/utmp` is normally writable only by root. In this event, the `ut_name` member must correspond to the actual user name associated with the process; the `ut_type` member must be either `USER_PROCESS` or `DEAD_PROCESS`; and the `ut_line` member must be a device special file and be writable by the user.

setutent() The **setutent()** function resets the input stream to the beginning of the file. This reset should be done before each search for a new entry if it is desired that the entire file be examined.

endutent() The **endutent()** function closes the currently open file.

utmpname() The **utmpname()** function allows the user to change the name of the file examined, from `/var/adm/utmp` to any other file. It is most often expected that this other file will be `/var/adm/wtmp`. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The **utmpname()** function does not open the file but closes the old file if it is currently open and saves the new file name.

RETURN VALUES

A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write. If the file name given is longer than 79 characters, **utmpname()** returns 0. Otherwise, it returns 1.

USAGE

These functions use buffered standard I/O for input, but **pututline()** uses an unbuffered non-standard write to avoid race conditions between processes trying to modify the `utmp` and `wtmp` files.

Applications should not access the `utmp` or `utmpx` database files directly, but should use the functions described on the **getutxent(3C)** manual page to interact with these files. Using these extended APIs will ensure that the `utmp` and `utmpx` databases are maintained consistently.

FILES

| | |
|-----------------------------|--|
| <code>/var/adm/utmp</code> | user access and accounting information (old format) |
| <code>/var/adm/utmpx</code> | user access and accounting information (new format) |
| <code>/var/adm/wtmp</code> | history of user access and accounting information (old format) |
| <code>/var/adm/wtmpx</code> | history of user access and accounting information (new format) |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

getutxent(3C) , **ttyslot(3C)** , **utmp(4)** , **utmpx(4)** , **attributes(5)**

NOTES

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either **getutid()** or **getutline()** , the function examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use **getutline()** to search for multiple occurrences, it would be necessary to zero out the static area after each success, or **getutline()** would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by **pututline()** (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the **getutent()** , **getutid()** or **getutline()** functions, if the user has just modified those contents and passed the pointer back to **pututline()** .

| | |
|--------------------|--|
| NAME | getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – access utmpx file entry |
| SYNOPSIS | <pre>#include <utmpx.h> struct utmpx * getutxent(void); struct utmpx * getutxid(const struct utmpx * <i>id</i>); struct utmpx * getutxline(const struct utmpx * <i>line</i>); struct utmpx * pututxline(const struct utmpx * <i>utmpx</i>); void setutxent(void); void endutxent(void); int utmpxname(const char * <i>file</i>); void getutmp(struct utmpx * <i>utmpx</i>, struct utmp * <i>utmp</i>); void getutmpx(struct utmp * <i>utmp</i>, struct utmpx * <i>utmpx</i>); void updwtmp(char * <i>wfile</i>, struct utmp * <i>utmp</i>); void updwtmpx(char * <i>wfilex</i>, struct utmpx * <i>utmpx</i>);</pre> |
| DESCRIPTION | <p>The getutxent() , getutxid() , and getutxline() functions each return a pointer to a utmpx structure with the following members:</p> <pre>char ut_user[32]; /* user login name */ char ut_id[4]; /* /etc/inittab id (usually line #) */ char ut_line[32]; /* device name (console, lnxx) */ pid_t ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ struct timeval ut_tv; /* time entry was made */ long ut_session; /* session ID, used for windowing */ long pad[5]; /* reserved for future use */ short ut_syslen; /* significant length of ut_host */ /* including terminating null */ char ut_host[257]; /* host name, if remote */</pre> <p>The structure exit status includes the following members:</p> <pre>short e_termination; /* termination status */ short e_exit; /* exit status */</pre> |
| getutxent() | The getutxent() function reads in the next entry from a utmpx -like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails. |

| | |
|---------------------|--|
| getutxid() | The getutxid() function searches forward from the current point in the <code>utmpx</code> file until it finds an entry with a <code>ut_type</code> matching <code>id</code> \Rightarrow <code>ut_type</code> , if the type specified is <code>RUN_LVL</code> , <code>BOOT_TIME</code> , <code>OLD_TIME</code> , or <code>NEW_TIME</code> . If the type specified in <code>id</code> is <code>INIT_PROCESS</code> , <code>LOGIN_PROCESS</code> , <code>USER_PROCESS</code> , or <code>DEAD_PROCESS</code> , then getutxid() will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <code>id</code> \Rightarrow <code>ut_id</code> . If the end of file is reached without a match, it fails. |
| getutxline() | The getutxline() function searches forward from the current point in the <code>utmpx</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>USER_PROCESS</code> which also has a <code>ut_line</code> string matching the <code>line</code> \Rightarrow <code>ut_line</code> string. If the end of file is reached without a match, it fails. |
| pututxline() | The pututxline() function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses getutxid() to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of pututxline() will have searched for the proper entry using one of the getutx() routines. If so, pututxline() will not search. If pututxline() does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the <code>utmpx</code> structure. When called by a non-root user, pututxline() invokes a setuid() root program to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by root. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either <code>USER_PROCESS</code> or <code>DEAD_PROCESS</code> ; and the <code>ut_line</code> member must be a device special file and be writable by the user. |
| setutxent() | The setutxent() function resets the input stream to the beginning of the file. This should be done before each search for a new entry if it is desired that the entire file be examined. |
| endutxent() | The endutxent() function closes the currently open file. |
| utmpxname() | The utmpxname() function allows the user to change the name of the file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code> . If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The utmpxname() function does not open the file, but closes the old file if it is currently open and saves the new file name. The new file name must end with the "f3x" character to allow the name of the corresponding <code>utmp</code> file to be easily obtainable; otherwise, an error code of 1 is returned. |

getutmp() The **getutmp()** function copies the information stored in the members of the `utmpx` structure to the corresponding members of the `utmp` structure. If the information in any member of `utmpx` does not fit in the corresponding `utmp` member, the data is truncated. (See **getutent(3C)** for `utmp` structure)

getutmpx() The **getutmpx()** function copies the information stored in the members of the `utmp` structure to the corresponding members of the `utmpx` structure. (See **getutent(3C)** for `utmp` structure)

updwtmp() The **updwtmp()** function checks the existence of `wfile` and its parallel file, whose name is obtained by appending an “f3x” to `wfile`. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. `utmp` is written to `wfile` and the corresponding `utmpx` structure is written to the parallel file.

updwtmpx() The **updwtmpx()** function checks the existence of `wfilex` and its parallel file, whose name is obtained by truncating the final “f3x” from `wfilex`. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. `utmpx` is written to `wfilex`, and the corresponding `utmp` structure is written to the parallel file.

RETURN VALUES

A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write.

USAGE

These functions use buffered standard I/O for input, but **pututxline()** uses an unbuffered write to avoid race conditions between processes trying to modify the `utmpx` and `wtmpx` files.

Applications should not access the `utmp` or `utmpx` database files directly, but should use the functions described on this manual page to interact with these files. Using these extended APIs will ensure that the `utmp` and `utmpx` databases are maintained consistently.

FILES

| | |
|-----------------------------|--|
| <code>/var/adm/utmp</code> | user access and accounting information (old format) |
| <code>/var/adm/utmpx</code> | user access and accounting information (new format) |
| <code>/var/adm/wtmp</code> | history of user access and accounting information (old format) |
| <code>/var/adm/wtmpx</code> | history of user access and accounting information (new format) |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

getutent(3C) , **ttyslot(3C)** , **utmp(4)** , **utmpx(4)** , **attributes(5)**

NOTES

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either **getutxid()** or **getutxline()** , the routine examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use **getutxline()** to search for multiple occurrences it would be necessary to zero out the static after each success, or **getutxline()** would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by **pututxline()** (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the **getutxent()** , **getutxid()** , or **getutxline()** routines, if the user has just modified those contents and passed the pointer back to **pututxline()** .

| | |
|----------------------|---|
| NAME | getvfsent, getvfile, getvfsspec, getvfsany – get vfstab file entry |
| SYNOPSIS | <pre>#include <stdio.h> #include <sys/vfstab.h> int getvfsent(FILE * fp, struct vfstab * vp); int getvfile(FILE * fp, struct vfstab * vp, char * file); int getvfsspec(FILE *, struct vfstab * vp, char * spec); int getvfsany(FILE *, struct vfstab * vp, struct vfstab * vref);</pre> |
| DESCRIPTION | <p>The getvfsent() , getvfile() , getvfsspec() , and getvfsany() functions each fill in the structure pointed to by <i>vp</i> with the broken-out fields of a line in the <code>/etc/vfstab</code> file. Each line in the file contains a <code>vfstab</code> structure, declared in the <code><sys/vfstab.h></code> header, whose following members are described on the vfstab(4) manual page:</p> <pre>\011char\011*vfs_special; \011char\011*vfs_fsckdev; \011char\011*vfs_mountp; \011char\011*vfs_fstype; \011char\011*vfs_fsckpass; \011char\011*vfs_automnt; \011char\011*vfs_mntopts;</pre> <p>The getvfsent() function returns a pointer to the next <code>vfstab</code> structure in the file; so successive calls can be used to search the entire file.</p> <p>The getvfile() function searches the file referenced by <i>fp</i> until a mount point matching <i>file</i> is found and fills <i>vp</i> with the fields from the line in the file.</p> <p>The getvfsspec() function searches the file referenced by <i>fp</i> until a special device matching <i>spec</i> is found and fills <i>vp</i> with the fields from the line in the file. The <i>spec</i> argument will try to match on device type (block or character special) and major and minor device numbers. If it cannot match in this manner, then it compares the strings.</p> <p>The getvfsany() function searches the file referenced by <i>fp</i> until a match is found between a line in the file and <i>vref</i> . A match occurs if all non-null entries in <i>vref</i> match the corresponding fields in the file.</p> <p>Note that these functions do not open, close, or rewind the file.</p> |
| RETURN VALUES | If the next entry is successfully read by getvfsent() or a match is found with getvfile() , getvfsspec() , or getvfsany() , 0 is returned. If an end-of-file is |

encountered on reading, these functions return `-1` . If an error is encountered, a value greater than `0` is returned. The possible error values are:

`VFS_TOOLONG` A line in the file exceeded the internal buffer size of `VFS_LINE_MAX` .

`VFS_TOOMANY` A line in the file contains too many fields.

`VFS_TOOFEW` A line in the file contains too few fields.

FILES

`/etc/vfstab`

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

`vfstab(4)` , `attributes(5)`

NOTES

The members of the `vfstab` structure point to information contained in a static area, so it must be copied if it is to be saved.

| NAME | getwc – get wide character from a stream | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <stdio.h> #include <wchar.h> wint_t getwc(FILE *stream);</pre> | | | | |
| DESCRIPTION | The getwc() function is equivalent to fgetwc(3S) , except that if it is implemented as a macro it may evaluate <i>stream</i> more than once, so the argument should never be an expression with side effects. | | | | |
| RETURN VALUES | Refer to fgetwc(3S) . | | | | |
| ERRORS | Refer to fgetwc(3S) . | | | | |
| USAGE | <p>This interface is provided to align with some current implementations and with possible future ISO standards.</p> <p>Because it may be implemented as a macro, getwc() may treat incorrectly a <i>stream</i> argument with side effects. In particular, getwc(*f++) may not work as expected. Therefore, use of this function is not recommended; fgetwc(3S) should be used instead.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | fgetwc(3S) , attributes(5) | | | | |

| | |
|--------------------|---|
| NAME | get_wch, wget_wch, mvget_wch, mvwget_wch – get a wide character from terminal |
| SYNOPSIS | <pre>#include <curses.h> int get_wch(wint_t * ch); int wget_wch(WINDOW * win, wint_t * ch); int mvget_wch(int y, int x, wint_t * ch); int mvwget_wch(WINDOW * win, int y, int x, wint_t * ch);</pre> |
| PARAMETERS | <p>ch Is a pointer to a wide integer where the returned wide character or KEY_ value can be stored.</p> <p>win Is a pointer to the window associated with the terminal from which the character is to be read.</p> <p>y Is the y (row) coordinate for the position of the character to be read.</p> <p>x Is the x (column) coordinate for the position of the character to be read.</p> |
| DESCRIPTION | <p>The get_wch() and wget_wch() functions get a wide character from the terminal associated with the window <code>stdscr</code> or window <code>win</code>, respectively. The mvget_wch() and mvwget_wch() functions move the cursor to the position specified in <code>stdscr</code> or <code>win</code>, respectively, then get a character.</p> <p>If the window is not a pad and has been changed since the last call to refresh(3XC), get_wch() calls refresh() to update the window before the next character is read.</p> <p>The setting of certain functions affects the behavior of the get_wch() set of functions. For example, if cbreak(3XC) is set, characters typed by the user are immediately processed. If halfdelay(3XC) is set, get_wch() waits until a character is typed or returns ERR if no character is typed within the specified timeout period. This timeout can also be specified for individual windows with the <code>delay</code> parameter of timeout(3XC). A negative value waits for input; a value of 0 returns ERR if no input is ready; a positive value blocks until input arrives or the time specified expires (in which case ERR is returned). If nodelay(3XC) is set, ERR is returned if no input is waiting; if not set, get_wch() waits until input arrives. Each character will be echoed to the window unless noecho(3XC) has been set.</p> <p>If keypad handling is enabled (keypad(3XC) is TRUE), the token for the function key (a KEY_ value) is stored in the object pointed to by <code>ch</code> and</p> |

`KEY_CODE_YES` is returned. If a character is received that could be the beginning of a function key (for example, `ESC`), an inter-byte timer is set. If the remainder of the sequence is not received before the time expires, the character is passed through; otherwise, the value of the function key is returned. If `notimeout()` is set, the inter-byte timer is not used.

The `ESC` key is typically a prefix key used with function keys and should not be used as a single character.

See the `getch(3XC)` manual page for a list of tokens for function keys that are returned by the `get_wch()` set of functions if keypad handling is enabled (Some terminals may not support all tokens).

RETURN VALUES

When these functions successfully report the pressing of a function key, they return `KEY_CODE_YES`. When they successfully report a wide character, they return `OK`. Otherwise, they return `ERR`.

ERRORS

None.

SEE ALSO

`cbreak(3XC)`, `echo(3XC)`, `halfdelay(3XC)`, `keypad(3XC)`, `nodelay(3XC)`, `notimeout(3XC)`, `raw(3XC)`, `timeout(3XC)`

| NAME | getwchar – get wide character from stdin stream | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | #include <wchar.h> wint_t getwchar(void); | | | | |
| DESCRIPTION | The getwchar() function is equivalent to <code>getwc(stdin)</code> . | | | | |
| RETURN VALUES | Refer to fgetwc(3S) . | | | | |
| ERRORS | Refer to fgetwc(3S) . | | | | |
| USAGE | If the <code>wint_t</code> value returned by getwchar() is stored into a variable of type <code>wchar_t</code> and then compared against the <code>wint_t</code> macro <code>WEOF</code> , the comparison may never succeed because <code>wchar_t</code> is defined as unsigned. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | fgetwc(3S) , getwc(3S) , attributes(5) | | | | |

| | |
|----------------------|--|
| NAME | getwd – get current working directory pathname |
| SYNOPSIS | <pre>#include <unistd.h> char *getwd(char *path_name);</pre> |
| DESCRIPTION | <p>The getwd() function determines an absolute pathname of the current working directory of the calling process, and copies that pathname into the array pointed to by the <i>path_name</i> argument.</p> <p>If the length of the pathname of the current working directory is greater than (<code>PATH_MAX + 1</code>) including the null byte, getwd() fails and returns a null pointer.</p> |
| RETURN VALUES | Upon successful completion, a pointer to the string containing the absolute pathname of the current working directory is returned. Otherwise, getwd() returns a null pointer and the contents of the array pointed to by <i>path_name</i> are undefined. |
| ERRORS | No errors are defined. |
| USAGE | For portability to implementations conforming to versions of the X/Open Portability Guide prior to SUS, getcwd(3C) is preferred over this function. |
| SEE ALSO | getcwd(3C) , standards(5) |

| NAME | getwidth – get codeset information | | | | |
|--------------------|---|----------------|-----------------|----------|-------------------------|
| SYNOPSIS | <pre>#include <euc.h> #include <getwidth.h> void getwidth(eucwidth_t *ptr);</pre> | | | | |
| DESCRIPTION | <p>The getwidth() function reads the character class table for the current locale to get information on the supplementary codesets. getwidth() sets this information into the struct <code>eucwidth_t</code>. This struct is defined in <code><euc.h></code> and has the following members:</p> <pre>short int _eucw1, _eucw2, _eucw3; short int _scrw1, _scrw2, _scrw3; short int _pcw; char _multibyte;</pre> <p>Codeset width values for supplementary codesets 1, 2, and 3 are set in <code>_eucw1</code>, <code>_eucw2</code>, and <code>_eucw3</code>, respectively. Screen width values for supplementary codesets 1, 2, and 3 are set in <code>_scrw1</code>, <code>_scrw2</code>, and <code>_scrw3</code>, respectively.</p> <p>The width of Extended Unix Code (EUC) Process Code is set in <code>_pcw</code>. The <code>_multibyte</code> entry is set to 1 if multibyte characters are used, and set to 0 if only single-byte characters are used.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe with exceptions</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe with exceptions |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe with exceptions | | | | |
| SEE ALSO | euclen(3C) , setlocale(3C) , attributes(5) | | | | |
| NOTES | <p>The getwidth() function can be used safely in a multithreaded application, as long as setlocale(3C) is not being called to change the locale.</p> <p>The getwidth() function will only work with EUC locales.</p> | | | | |

| | |
|----------------------|---|
| NAME | getwin, putwin – read a window from, and write a window to, a file |
| SYNOPSIS | <pre>#include <curses.h> WINDOW * getwin(FILE * filep); int putwin(WINDOW * win, FILE * filep);</pre> |
| PARAMETERS | <p>filep Is a pointer to a <code>stdio</code> stream.</p> <p>win Is a pointer to a window.</p> |
| DESCRIPTION | <p>The getwin() function reads window-related data (written earlier by putwin()) from the <code>stdio</code> stream pointed to by <i>filep</i> . It then creates and initializes a new window using that data.</p> <p>The putwin() function writes all the data associated with the window pointed to by <i>win</i> to the <code>stdio</code> stream pointed to by <i>filep</i> . The getwin() function can later retrieve this data.</p> |
| RETURN VALUES | <p>On success, the getwin() function returns a pointer to the new window created. Otherwise, it returns a null pointer.</p> <p>On success, the putwin() function returns <code>OK</code> . Otherwise, it returns <code>ERR</code> .</p> |
| ERRORS | None. |
| SEE ALSO | <code>scr_dump(3XC)</code> |

| NAME | getws, fgetws – convert a string of EUC characters from the stream to Process Code | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <stdio.h> #include <wdec.h> wchar_t * getws(wchar_t * s); #include <stdio.h> #include <wchar.h> wchar_t * fgetws(wchar_t * s, int n, FILE* stream);</pre> | | | | |
| DESCRIPTION | <p>The getws() function reads a string of Extended Unix Code (EUC) characters from the standard input stream, <code>stdin</code>, converts it to process code, and writes it to the array pointed to by <code>s</code>, until a new-line character is read or an end-of-file condition is encountered. The newline character is discarded and the string is terminated with a <code>wchar_t</code> null character. The getws() function returns its argument.</p> <p>The fgetws() function reads EUC characters from the <code>stream</code>, converts them to Process Code, and writes them to the array pointed to by <code>s</code>. It stops when either <code>n - 1</code> characters are read, a newline character is read and transferred to <code>s</code>, or an end-of-file condition is encountered. The string is then terminated with a <code>wchar_t</code> null character. The fgetws() function returns its first argument.</p> | | | | |
| RETURN VALUES | If end-of-file is encountered and no characters have been read, no characters are transferred to <code>s</code> and a null pointer is returned. If a read error occurs, such as trying to use these functions on a file that has not been opened for reading, a null pointer is returned. Otherwise, <code>s</code> is returned. | | | | |
| ERRORS | <p>The fgetws() function will fail if data needs to be read and:</p> <p>EOverflow The file is a regular file and an attempt was made to read at or beyond the offset maximum associated with the corresponding <code>stream</code>.</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | ferror(3S) , fread(3S) , getwc(3S) , putws(3S) , scanf(3S) , attributes(5) | | | | |

| | |
|--------------------------------|---|
| NAME | glob, globfree – generate path names matching a pattern |
| SYNOPSIS | <pre>#include <glob.h> int glob(const char * pattern, int flags, int(* errfunc)(const char * epath int eerrno), glob_t * pglob); void globfree(glob_t * pglob);</pre> |
| DESCRIPTION | <p>The glob() function is a path name generator.</p> <p>The globfree() function frees any memory allocated by glob() associated with <i>pglob</i> .</p> |
| <i>pattern</i> Argument | <p>The argument <i>pattern</i> is a pointer to a path name pattern to be expanded. The glob() function matches all accessible path names against this pattern and develops a list of all path names that match. In order to have access to a path name, glob() requires search permission on every component of a path except the last, and read permission on each directory of any filename component of <i>pattern</i> that contains any of the following special characters:</p> <pre>* ? [</pre> |
| <i>pglob</i> Argument | <p>The structure type <code>glob_t</code> is defined in the header <code><glob.h></code> and includes at least the following members:</p> <pre>size_t gl_pathc; /* count of paths matched by pattern */ char **gl_pathv; /* pointer to list of matched path names */ size_t gl_offs; /* slots to reserve at beginning of gl_pathv */</pre> <p>The glob() function stores the number of matched path names into <i>pglob</i>→<code>gl_pathc</code> and a pointer to a list of pointers to path names into <i>pglob</i>→<code>gl_pathv</code>. The path names are in sort order as defined by the current setting of the LC_COLLATE category. The first pointer after the last path name is a NULL pointer. If the pattern does not match any path names, the returned number of matched paths is set to 0, and the contents of <i>pglob</i>→<code>gl_pathv</code> are implementation-dependent.</p> <p>It is the caller's responsibility to create the structure pointed to by <i>pglob</i> . The glob() function allocates other space as needed, including the memory pointed to by <code>gl_pathv</code> . The globfree() function frees any space associated with <i>pglob</i> from a previous call to glob() .</p> |
| <i>flags</i> Argument | <p>The <i>flags</i> argument is used to control the behavior of glob() . The value of <i>flags</i> is a bitwise inclusive OR of zero or more of the following constants, which are defined in the header <code><glob.h></code> :</p> |

| | |
|---------------|---|
| GLOB_APPEND | Append path names generated to the ones from a previous call to glob() . |
| GLOB_DOOFFS | Make use of <i>pglob->gl_offs</i> . If this flag is set, <i>pglob->gl_offs</i> is used to specify how many NULL pointers to add to the beginning of <i>pglob->gl_pathv</i> . In other words, <i>pglob->gl_pathv</i> will point to <i>pglob->gl_offs</i> NULL pointers, followed by <i>pglob->gl_pathc</i> path name pointers, followed by a NULL pointer. |
| GLOB_ERR | Causes glob() to return when it encounters a directory that it cannot open or read. Ordinarily, glob() continues to find matches. |
| GLOB_MARK | Each path name that is a directory that matches <i>pattern</i> has a slash appended. |
| GLOB_NOCHECK | If <i>pattern</i> does not match any path name, then glob() returns a list consisting of only <i>pattern</i> , and the number of matched path names is 1. |
| GLOB_NOESCAPE | Disable backslash escaping. |
| GLOB_NOSORT | Ordinarily, glob() sorts the matching path names according to the current setting of the LC_COLLATE category. When this flag is used the order of path names returned is unspecified. |

The GLOB_APPEND flag can be used to append a new set of path names to those found in a previous call to **glob()** . The following rules apply when two or more calls to **glob()** are made with the same value of *pglob* and without intervening calls to **globfree()** :

1. The first such call must not set GLOB_APPEND . All subsequent calls must set it.
2. All the calls must set GLOB_DOOFFS , or all must not set it.
3. After the second call, *pglob->gl_pathv* points to a list containing the following:
 - a. Zero or more NULL pointers, as specified by GLOB_DOOFFS and *pglob->gl_offs* .
 - b. Pointers to the path names that were in the *pglob->gl_pathv* list before the call, in the same order as before.
 - c. Pointers to the new path names generated by the second call, in the specified order.

4. The count returned in *pglob*-> *gl_pathc* will be the total number of path names from the two calls.
5. The application can change any of the fields after a call to **glob()** . If it does, it must reset them to the original value before a subsequent call, using the same *pglob* value, to **globfree()** or **glob()** with the `GLOB_APPEND` flag.

***errfunc* and *epath* Arguments**

If, during the search, a directory is encountered that cannot be opened or read and *errfunc* is not a NULL pointer, **glob()** calls (**errfunc*) with two arguments:

1. The *epath* argument is a pointer to the path that failed.
2. The *errno* argument is the value of *errno* from the failure, as set by the `opendir(3C)` , `readdir(3C)` or `stat(2)` functions. (Other values may be used to report other errors not explicitly documented for those functions.)

The following constants are defined as error return values for **glob()** :

`GLOB_ABORTED` The scan was stopped because `GLOB_ERR` was set or (**errfunc*) returned non-zero.

`GLOB_NOMATCH` The pattern does not match any existing path name, and `GLOB_NOCHECK` was not set in flags.

`GLOB_NOSPACE` An attempt to allocate memory failed.

If (**errfunc*) is called and returns non-zero, or if the `GLOB_ERR` flag is set in *flags* , **glob()** stops the scan and returns `GLOB_ABORTED` after setting *gl_pathc* and *gl_pathv* in *pglob* to reflect the paths already scanned. If `GLOB_ERR` is not set and either *errfunc* is a NULL pointer or (**errfunc*) returns 0, the error is ignored.

RETURN VALUES

The following values are returned by **glob()** :

0 Successful completion. The argument *pglob*-> *gl_pathc* returns the number of matched path names and the argument *pglob*-> *gl_pathv* contains a pointer to a null-terminated list of matched and sorted path names. However, if *pglob*-> *gl_pathc* is 0, the content of *pglob*-> *gl_pathv* is undefined.

non-zero An error has occurred. Non-zero constants are defined in <glob.h> . The arguments *pglob*-> *gl_pathc* and *pglob*-> *gl_pathv* are still set as defined above.

The **globfree()** function returns no value.

USAGE

This function is not provided for the purpose of enabling utilities to perform path name expansion on their arguments, as this operation is performed by the shell, and utilities are explicitly not expected to redo this. Instead, it is provided

for applications that need to do path name expansion on strings obtained from other sources, such as a pattern typed by a user or read from a file.

If a utility needs to see if a path name matches a given pattern, it can use `fnmatch(3C)`.

Note that `gl_pathc` and `gl_pathv` have meaning even if `glob()` fails. This allows `glob()` to report partial results in the event of an error. However, if `gl_pathc` is 0, `gl_pathv` is unspecified even if `glob()` did not return an error.

The `GLOB_NOCHECK` option could be used when an application wants to expand a path name if wildcards are specified, but wants to treat the pattern as just a string otherwise.

The new path names generated by a subsequent call with `GLOB_APPEND` are not sorted together with the previous path names. This mirrors the way that the shell handles path name expansion when multiple expansions are done on a command line.

Applications that need tilde and parameter expansion should use the `wordexp(3C)` function.

EXAMPLES

EXAMPLE 1 Example of `glob_dooofs` function.

One use of the `GLOB_DOOFFS` flag is by applications that build an argument list for use with the `execv()`, `execve()`, or `execvp()` functions (see `exec(2)`). Suppose, for example, that an application wants to do the equivalent of:

```
ls
-l
*.c
```

but for some reason:

```
system("ls -l *.c")
```

is not acceptable. The application could obtain approximately the same result using the sequence:

```
gloffset.gl_offs = 2;
glob ("*.c", GLOB_DOOFFS, NULL, &gloffset);
gloffset.gl_pathv[0] = "ls";
gloffset.gl_pathv[1] = "-l";
execvp ("ls", &gloffset.gl_pathv[0]);
```

Using the same example:

```
ls
-l
*.c *.h
```

could be approximately simulated using GLOB_APPEND as follows:

```
globbuf.gl_offs = 2;
glob ("*.c", GLOB_DOOFFS, NULL, &globbuf);
glob ("*.h", GLOB_DOOFFS|GLOB_APPEND, NULL, &globbuf);
...
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

execv(2), **stat(2)**, **fnmatch(3C)**, **opendir(3C)**, **readdir(3C)**, **wordexp(3C)**, **attributes(5)**

NAME global_variables – variables used for X/Open Curses

DESCRIPTION The global variables defined for X/Open Curses are as follows:
Definitions of Global Variables

| Constant | Description |
|--------------|--|
| COLORS | Number of colors supported by terminal |
| COLOR_PAIRS | Number of color pairs supported by terminal |
| COLS | Number of columns supported by terminal |
| LINES | Number of lines supported by terminal |
| boolcodes[] | termcap capability names |
| boolfnames[] | Full C names |
| boolnames[] | terminfo capability names |
| cur_term | Current terminal |
| curscr | Current screen image |
| numcodes[] | termcap capability codes |
| numfnames[] | Full C names |
| numnames[] | terminfo capability codes |
| stdscr | Standard screen supplied by <code>initscr()</code> |
| strcodes[] | termcap capability name |
| strfnames[] | Full C names |
| strnames[] | terminfo capability names |
| ttytype | Terminal type |

The `boolcodes[]`, `boolfnames[]`, `boolnames[]`, `numcodes[]`, `numfnames[]`, `numnames[]`, `strcodes[]`, `strfnames[]`, `strnames[]`, and `ttytype` constants conform to UNIX System V.

The `curscr`, `stdscr`, `cur_term`, `COLS`, `LINES`, `COLORS`, and `COLOR_PAIRS`, constants conform to UNIX System V and XPG4 version 2.

NAME gmatch – shell global pattern matching

SYNOPSIS `cc [flag ...] file ... -lgen [library ...]`
`#include <libgen.h>`

`int gmatch(const char *str, const char *pattern);`

DESCRIPTION **gmatch()** checks whether the null-terminated string *str* matches the null-terminated pattern string *pattern*. See the **sh(1)**, section File Name Generation, for a discussion of pattern matching. A backslash (\) is used as an escape character in pattern strings.

RETURN VALUES **gmatch()** returns non-zero if the pattern matches the string, zero if the pattern does not.

EXAMPLES **EXAMPLE 1** Examples of **gmatch()** function.

In the following example, **gmatch()** returns non-zero (true) for all strings with “a” or “-” as their last character.

```
char *s;
gmatch (s, "[a\-" )
```

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **sh(1)**, **attributes(5)**

NOTES When compiling multithreaded applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multithreaded applications.

| NAME | grantpt – grant access to the slave pseudo-terminal device | | | | |
|----------------------|--|----------------|-----------------|----------|------|
| SYNOPSIS | #include <stdlib.h> int grantpt(int <i>fildev</i>); | | | | |
| DESCRIPTION | The grantpt() function changes the mode and ownership of the slave pseudo-terminal device associated with its master pseudo-terminal counter part. <i>fildev</i> is the file descriptor returned from a successful open of the master pseudo-terminal device. A <i>setuid</i> root program (see setuid(2)) is invoked to change the permissions. The user ID of the slave is set to the real UID of the calling process and the group ID is set to a reserved group. The permission mode of the slave pseudo-terminal is set to readable and writable by the owner and writable by the group. | | | | |
| RETURN VALUES | Upon successful completion, grantpt() returns 0. Otherwise, it returns -1 and sets <i>errno</i> to indicate the error. | | | | |
| ERRORS | The grantpt() function may fail if: EBADF The <i>fildev</i> argument is not a valid open file descriptor. EINVAL The <i>fildev</i> argument is not associated with a master pseudo-terminal device. EACCES The corresponding slave pseudo-terminal device could not be accessed. | | | | |
| USAGE | The grantpt() function will fail if it is unable to successfully invoke the <i>setuid</i> root program. It may also fail if the application has installed a signal handler to catch SIGCHLD signals. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe | | | | |
| SEE ALSO | open(2) , setuid(2) , ptsname(3C) , unlockpt(3C) , attributes(5) <i>STREAMS Programming Guide</i> | | | | |

| | |
|----------------------|--|
| NAME | halfdelay – enable/disable half-delay mode |
| SYNOPSIS | <pre>#include <curses.h> int halfdelay(int tenths);</pre> |
| PARAMETERS | <p><i>tenths</i> Is the number of tenths of seconds for which to block input (1 to 255).</p> |
| DESCRIPTION | <p>The halfdelay() function is similar to cbreak(3XC) in that when set, characters typed by the user are immediately processed by the program. The difference is that ERR is returned if no input is received after <i>tenths</i> tenths seconds.</p> <p>The nocbreak(3XC) function should be used to leave half-delay mode.</p> |
| RETURN VALUES | On success, the halfdelay() function returns OK . Otherwise, it returns ERR . |
| ERRORS | None. |
| SEE ALSO | cbreak(3XC) |

| | |
|----------------------|--|
| NAME | has_ic, has_il – determine insert/delete character/line capability |
| SYNOPSIS | <pre>#include <curses.h> bool has_ic(void); bool has_il(void);</pre> |
| DESCRIPTION | <p>The has_ic() function determines whether or not the terminal has insert/delete character capability.</p> <p>The has_il() function determines whether or not the terminal has insert/delete line capability.</p> |
| RETURN VALUES | <p>The has_ic() function returns <code>TRUE</code> if the terminal has insert/delete character capability and <code>FALSE</code> otherwise.</p> <p>The has_il() function returns <code>TRUE</code> if the terminal has insert/delete line capability and <code>FALSE</code> otherwise.</p> |
| ERRORS | None. |

| | |
|--------------------|---|
| NAME | hline, mvhline, mvvline, mvwhline, mvwvline, vline, whline, wvline – use single-byte characters (and renditions) to draw lines |
| SYNOPSIS | <pre>#include <curses.h> int hline(chtype <i>ch</i>, int <i>n</i>); int mvhline(int <i>y</i>, int <i>x</i>, chtype <i>ch</i>, int <i>n</i>); int mvvline(int <i>y</i>, int <i>x</i>, chtype <i>ch</i>, int <i>n</i>); int mvwhline(WINDOW * <i>win</i>, int <i>y</i>, int <i>x</i>, chtype <i>ch</i>, int <i>n</i>); int mvwvline(WINDOW * <i>win</i>, int <i>y</i>, int <i>x</i>, chtype <i>ch</i>, int <i>n</i>); int vline(chtype <i>ch</i>, int <i>n</i>); int whline(WINDOW * <i>win</i>, chtype <i>ch</i>, int <i>n</i>); int wvline(WINDOW * <i>win</i>, chtype <i>ch</i>, int <i>n</i>);</pre> |
| PARAMETERS | <p><i>ch</i> Is the character used to draw the line.</p> <p><i>n</i> Is the maximum number of characters in the line.</p> <p><i>y</i> Is the y (row) coordinate for the start of the line.</p> <p><i>x</i> Is the x (column) coordinate for the start of the line.</p> <p><i>win</i> Is a pointer to a window.</p> |
| DESCRIPTION | <p>The hline() , vline() , whline() , wvline() functions draw a horizontal or vertical line, in either the window <code>stdscr</code> or <i>win</i> starting at the current cursor position. The line is drawn using the character <i>ch</i> and is a maximum of <i>n</i> positions long, or as many as will fit into the window. If <i>ch</i> is 0 (zero), the default horizontal or vertical character is used.</p> <p>The mvhline() , mvvline() , mvwhline() , mvwvline() functions are similar to the previous group of functions but the line begins at cursor position specified by <i>x</i> and <i>y</i> .</p> <p>The functions with names ending with hline() draw horizontal lines proceeding towards the last column of the same line. The functions with names ending with vline() draw vertical lines proceeding towards the last column of the same line.</p> <p>These functions do not change the position of the cursor.</p> |

| | |
|----------------------|--|
| RETURN VALUES | On success, these functions return <code>OK</code> . Otherwise, they return <code>ERR</code> . |
| ERRORS | None |
| SEE ALSO | <code>border(3XC)</code> , <code>border_set(3XC)</code> , <code>hline_set(3XC)</code> |

| | | | | | | | | | | | |
|--------------------|--|------------|---|----------|--|----------|--|----------|---|------------|---------------------------|
| NAME | hline_set, mvhline_set, mvvline_set, mvwhline_set, mvwvline_set, vline_set, whline_set, wvline_set – use complex characters (and renditions) to draw lines | | | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int hline_set(const cchar_t * ch, int n); int mvhline_set(int y, int x, const cchar_t * wch, int n); int mvvline_set(int y, int x, const cchar_t * wch, int n); int mvwhline_set(WINDOW * win, int y, int x, const cchar_t * wch, int n); int mvwvline_set(WINDOW * win, int y, int x, const cchar_t * wch, int n); int vline_set(const cchar_t * wch, int n); int whline_set(WINDOW * win, const cchar_t * wch, int n); int wvline_set(WINDOW * win, const cchar_t * wch, int n);</pre> | | | | | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="padding-right: 20px;">wch</td> <td>Is the complex character used to draw the line.</td> </tr> <tr> <td>n</td> <td>Is the maximum number of characters in the line.</td> </tr> <tr> <td>y</td> <td>Is the y (row) coordinate for the start of the line.</td> </tr> <tr> <td>x</td> <td>Is the x (column) coordinate for the start of the line.</td> </tr> <tr> <td>win</td> <td>Is a pointer to a window.</td> </tr> </table> | wch | Is the complex character used to draw the line. | n | Is the maximum number of characters in the line. | y | Is the y (row) coordinate for the start of the line. | x | Is the x (column) coordinate for the start of the line. | win | Is a pointer to a window. |
| wch | Is the complex character used to draw the line. | | | | | | | | | | |
| n | Is the maximum number of characters in the line. | | | | | | | | | | |
| y | Is the y (row) coordinate for the start of the line. | | | | | | | | | | |
| x | Is the x (column) coordinate for the start of the line. | | | | | | | | | | |
| win | Is a pointer to a window. | | | | | | | | | | |
| DESCRIPTION | <p>The hline_set() , vline_set() , whline_set() , wvline_set() functions draw a line, in either the window <code>stdscr</code> or <code>win</code> starting at the current cursor position. The line is drawn using the character <code>wch</code> and is a maximum of <code>n</code> positions long, or as many as will fit into the window. If <code>wch</code> is a null pointer, the default horizontal or vertical character is used.</p> <p>The mvhline_set() , mvvline_set() , mvwhline_set() , mvwvline_set() functions are similar to the previous group of functions but the line begins at cursor position specified by <code>x</code> and <code>y</code> .</p> <p>The functions with names ending with hline_set() draw horizontal lines proceeding towards the last column of the same line. The functions with names ending with vline_set() draw vertical lines proceeding towards the last column of the same line.</p> <p>These functions do not change the position of the cursor.</p> | | | | | | | | | | |

RETURN VALUES

On success, these functions return `OK` . Otherwise, they return `ERR` .

ERRORS

None.

SEE ALSO

`border(3XC)` , `border_set(3XC)` , `hline(3XC)`

| | |
|----------------------|--|
| NAME | hsearch, hcreate, hdestroy – manage hash search tables |
| SYNOPSIS | <pre>#include <search.h> ENTRY * hsearch(ENTRY item, ACTION action); int hcreate(size_t mekments); void hdestroy(void);</pre> |
| DESCRIPTION | <p>The hsearch() function is a hash-table search routine generalized from Knuth (6.4) Algorithm D. It returns a pointer into a hash table indicating the location at which an entry can be found. The comparison function used by hsearch() is strcmp() (see string(3C)). The <i>item</i> argument is a structure of type ENTRY (defined in the <code><search.h></code> header) containing two pointers: <i>item.key</i> points to the comparison key, and <i>item.data</i> points to any other data to be associated with that key. (Pointers to types other than void should be cast to pointer-to-void.) The <i>action</i> argument is a member of an enumeration type ACTION (defined in <code><search.h></code>) indicating the disposition of the entry if it cannot be found in the table. ENTER indicates that the item should be inserted in the table at an appropriate point. Given a duplicate of an existing item, the new item is not entered and hsearch() returns a pointer to the existing item. FIND indicates that no entry should be made. Unsuccessful resolution is indicated by the return of a null pointer.</p> <p>The hcreate() function allocates sufficient space for the table, and must be called before hsearch() is used. The <i>nel</i> argument is an estimate of the maximum number of entries that the table will contain. This number may be adjusted upward by the algorithm in order to obtain certain mathematically favorable circumstances.</p> <p>The hdestroy() function destroys the search table, and may be followed by another call to hcreate().</p> |
| RETURN VALUES | <p>The hsearch() function returns a null pointer if either the action is FIND and the item could not be found or the action is ENTER and the table is full.</p> <p>The hcreate() function returns 0 if it cannot allocate sufficient space for the table.</p> |
| USAGE | <p>The hsearch() and hcreate() functions use malloc(3C) to allocate space.</p> <p>Only one hash search table may be active at any given time.</p> |
| EXAMPLES | <p>EXAMPLE 1 Example to read in strings.</p> <p>The following example will read in strings followed by two numbers and store them in a hash table, discarding duplicates. It will then read in strings and find the matching entry in the hash table and print it.</p> |

```

#include <stdio.h>
#include <search.h>
#include <string.h>
#include <stdlib.h>

struct info {
    int age, room;
};
#define NUM_EMPL 5000
main( )
{
    /* space to store strings */
    char string_space[NUM_EMPL*20];
    /* space to store employee info */
    struct info info_space[NUM_EMPL];
    /* next avail space in string_space */
    char *str_ptr = string_space;
    /* next avail space in info_space */
    struct info *info_ptr = info_space;
    ENTRY item, *found_item;
    /* name to look for in table */
    char name_to_find[30];
    int i = 0;

    /* create table */
    (void) hcreate(NUM_EMPL);
    while (scanf("%s%d%d", str_ptr, &info_ptr->age,
        &info_ptr->room) != EOF && i++ < NUM_EMPL) {
        /* put info in structure, and structure in item */
        item.key = str_ptr;
        item.data = (void *)info_ptr;
        str_ptr += strlen(str_ptr) + 1;
        info_ptr++;
        /* put item into table */
        (void) hsearch(item, ENTER);
    }

    /* access table */
    item.key = name_to_find;
    while (scanf("%s", item.key) != EOF) {
        if ((found_item = hsearch(item, FIND)) != NULL) {
            /* if item is in the table */
            (void)printf("found %s, age = %d, room = %d\n",
                found_item->key,
                ((struct info *)found_item->data)->age,
                ((struct info *)found_item->data)->room);
        } else {
            (void)printf("no such employee %s\n",
                name_to_find);
        }
    }
    return 0;
}

```

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

`bsearch(3C)`, `lsearch(3C)`, `malloc(3C)`, `string(3C)`, `tsearch(3C)`, `malloc(3X)`, `attributes(5)`

The Art of Computer Programming, Volume 3, Sorting and Searching by Donald E. Knuth, published by Addison-Wesley Publishing Company, 1973.

| | |
|----------------------|--|
| NAME | htonl, htons, ntohl, ntohs – convert values between host and network byte order |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lxnet [<i>library</i> ...] #include <arpa/inet.h> uint32_t htonl(uint32_t <i>hostlong</i>); uint16_t htons(uint16_t <i>hostshort</i>); uint32_t ntohl(uint32_t <i>netlong</i>); uint16_t ntohs(uint16_t <i>netshort</i>);</pre> |
| DESCRIPTION | <p>These functions convert 16-bit and 32-bit quantities between network byte order and host byte order.</p> <p>The <code>uint32_t</code> and <code>uint16_t</code> types are made available by inclusion of <code><inttypes.h></code>.</p> |
| USAGE | <p>These functions are most often used in conjunction with Internet addresses and ports as returned by <code>gethostent(3XN)</code> and <code>getservent(3XN)</code>.</p> <p>On some architectures these functions are defined as macros that expand to the value of their argument.</p> |
| RETURN VALUES | <p>The <code>htonl()</code> and <code>htons()</code> functions return the argument value converted from host to network byte order.</p> <p>The <code>ntohl()</code> and <code>ntohs()</code> functions return the argument value converted from network to host byte order.</p> |
| ERRORS | No errors are defined. |
| ATTRIBUTES | See <code>attributes(5)</code> for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO`endhostent(3XN)` , `endservent(3XN)` , `attributes(5)`

| NAME | hypot - Euclidean distance function | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | cc [<i>flag ...</i>] <i>file ...</i> -lm [<i>library ...</i>] #include <math.h> double hypot(double <i>x</i> , double <i>y</i>); | | | | |
| DESCRIPTION | The hypot() function computes the length of the hypotenuse of a right-angled triangle: <pre> 3 3 5 4 4 4 </pre> | | | | |
| RETURN VALUES | Upon successful completion, hypot() returns the length of the hypotenuse of a right angled triangle with sides of length <i>x</i> and <i>y</i> . If the result would cause overflow, HUGE_VAL is returned and <i>errno</i> may be set to ERANGE. If <i>x</i> or <i>y</i> is NaN, NaN is returned. | | | | |
| ERRORS | The hypot() function may fail if: ERANGE The result overflows. | | | | |
| USAGE | The hypot() function takes precautions against underflow and overflow during intermediate steps of the computation. An application wishing to check for error situations should set <i>errno</i> to 0 before calling hypot() . If <i>errno</i> is non-zero on return, or the return value is HUGE_VAL or NaN, an error has occurred. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: <table border="1" data-bbox="500 1329 1396 1413"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | isnan(3M) , sqrt(3M) , attributes(5) | | | | |

| | |
|--------------------|--|
| NAME | iconv – code conversion function |
| SYNOPSIS | <pre>#include<iconv.h> size_t iconv(iconv_t cd, const char **inbuf, size_t *inbytesleft, char **outbuf, size_t *outbytesleft);</pre> |
| DESCRIPTION | <p>The iconv() function converts the sequence of characters from one code set, in the array specified by <i>inbuf</i>, into a sequence of corresponding characters in another code set, in the array specified by <i>outbuf</i>. The code sets are those specified in the <i>iconv_open()</i> call that returned the conversion descriptor, <i>cd</i>. The <i>inbuf</i> argument points to a variable that points to the first character in the input buffer and <i>inbytesleft</i> indicates the number of bytes to the end of the buffer to be converted. The <i>outbuf</i> argument points to a variable that points to the first available byte in the output buffer and <i>outbytesleft</i> indicates the number of the available bytes to the end of the buffer.</p> <p>For state-dependent encodings, the conversion descriptor <i>cd</i> is placed into its initial shift state by a call for which <i>inbuf</i> is a null pointer, or for which <i>inbuf</i> points to a null pointer. When iconv() is called in this way, and if <i>outbuf</i> is not a null pointer or a pointer to a null pointer, and <i>outbytesleft</i> points to a positive value, iconv() will place, into the output buffer, the byte sequence to change the output buffer to its initial shift state. If the output buffer is not large enough to hold the entire reset sequence, iconv() will fail and set <i>errno</i> to <i>E2BIG</i>. Subsequent calls with <i>inbuf</i> as other than a null pointer or a pointer to a null pointer cause the conversion to take place from the current state of the conversion descriptor.</p> <p>If a sequence of input bytes does not form a valid character in the specified code set, conversion stops after the previous successfully converted character. If the input buffer ends with an incomplete character or shift sequence, conversion stops after the previous successfully converted bytes. If the output buffer is not large enough to hold the entire converted input, conversion stops just prior to the input bytes that would cause the output buffer to overflow. The variable pointed to by <i>inbuf</i> is updated to point to the byte following the last byte successfully used in the conversion. The value pointed to by <i>inbytesleft</i> is decremented to reflect the number of bytes still not converted in the input buffer. The variable pointed to by <i>outbuf</i> is updated to point to the byte following the last byte of converted output data. The value pointed to by <i>outbytesleft</i> is decremented to reflect the number of bytes still available in the output buffer. For state-dependent encodings, the conversion descriptor is updated to reflect the shift state in effect at the end of the last successfully converted byte sequence.</p> <p>If iconv() encounters a character in the input buffer that is legal, but for which an identical character does not exist in the target code set, iconv() performs an implementation-defined conversion on this character.</p> |

RETURN VALUES

The **iconv()** function updates the variables pointed to by the arguments to reflect the extent of the conversion and returns the number of non-identical conversions performed. If the entire string in the input buffer is converted, the value pointed to by *inbytesleft* will be 0. If the input conversion is stopped due to any conditions mentioned above, the value pointed to by *inbytesleft* will be non-zero and *errno* is set to indicate the condition. If an error occurs **iconv()** returns (*size_t*) -1 and sets *errno* to indicate the error.

ERRORS

The **iconv()** function will fail if:

- EILSEQ** Input conversion stopped due to an input byte that does not belong to the input code set.
- E2BIG** Input conversion stopped due to lack of space in the output buffer.
- EINVAL** Input conversion stopped due to an incomplete character or shift sequence at the end of the input buffer.

The **iconv()** function may fail if:

- EBADF** The *cd* argument is not a valid open conversion descriptor.

EXAMPLES**CODE EXAMPLE 1** Using the **iconv()** Functions

The following example uses the **iconv()** functions:

```
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <iconv.h>
#include <stdlib.h>

/*
 * For state-dependent encodings, changes the state of the conversion
 * descriptor to initial shift state. Also, outputs the byte sequence
 * to change the state to initial state.
 * This code is assuming the iconv call for initializing the state
 * won't fail due to lack of space in the output buffer.
 */
#define INIT_SHIFT_STATE(cd, fptr, ileft, tptr, oleft) \
{
    fptr = NULL; \
    ileft = 0; \
    tptr = to; \
    oleft = BUFSIZ; \
    (void) iconv(cd, &fptr, &ileft, &tptr, &oleft); \
    (void) fwrite(to, 1, BUFSIZ - oleft, stdout); \
}

int
main(int argc, char **argv)
{
    iconv_t cd;
```

```

char    from[BUFSIZ], to[BUFSIZ];
char    *from_code, *to_code;
char    *tptr;
const char *fptr;
size_t  ileft, oleft, num, ret;

if (argc != 3) {
    (void) fprintf(stderr,
        "Usage: %s from_codeset to_codeset\\n", argv[0]);
    return (1);
}

from_code = argv[1];
to_code = argv[2];

cd = iconv_open((const char *)to_code, (const char *)from_code);
if (cd == (iconv_t)-1) {
    /*
     * iconv_open failed
     */
    (void) fprintf(stderr,
        "iconv_open(%s, %s) failed\\n", to_code, from_code);
    return (1);
}

ileft = 0;
while ((ileft +=
    (num = fread(from + ileft, 1, BUFSIZ - ileft, stdin))) > 0) {
    if (num == 0) {
        /*
         * Input buffer still contains incomplete character
         * or sequence.  However, no more input character.
         */

        /*
         * Initializes the conversion descriptor and outputs
         * the sequence to change the state to initial state.
         */
        INIT_SHIFT_STATE(cd, fptr, ileft, tptr, oleft);
        (void) iconv_close(cd);

        (void) fprintf(stderr, "Conversion error\\n");
        return (1);
    }

    fptr = from;
    for (;;) {
        tptr = to;
        oleft = BUFSIZ;

        ret = iconv(cd, &fptr, &ileft, &tptr, &oleft);
        if (ret != (size_t)-1) {
            /*
             * iconv succeeded
             */

```

```
    /*
     * Outputs converted characters
     */
    (void) fwrite(to, 1, BUFSIZ - oleft, stdout);
    break;
}

/*
 * iconv failed
 */
if (errno == EINVAL) {
    /*
     * Incomplete character or shift sequence
     */

    /*
     * Outputs converted characters
     */
    (void) fwrite(to, 1, BUFSIZ - oleft, stdout);
    /*
     * Copies remaining characters in input buffer
     * to the top of the input buffer.
     */
    (void) memmove(from, fptr, ileft);
    /*
     * Tries to fill input buffer from stdin
     */
    break;
} else if (errno == E2BIG) {
    /*
     * Lack of space in output buffer
     */

    /*
     * Outputs converted characters
     */
    (void) fwrite(to, 1, BUFSIZ - oleft, stdout);
    /*
     * Tries to convert remaining characters in
     * input buffer with emptied output buffer
     */
    continue;
} else if (errno == EILSEQ) {
    /*
     * Illegal character or shift sequence
     */

    /*
     * Outputs converted characters
     */
    (void) fwrite(to, 1, BUFSIZ - oleft, stdout);
    /*
     * Initializes the conversion descriptor and
     * outputs the sequence to change the state to
     * initial state.
     */
    INIT_SHIFT_STATE(cd, fptr, ileft, tptr, oleft);
}
```

```

        (void) iconv_close(cd);

        (void) fprintf(stderr,
            "Illegal character or sequence\\n");
        return (1);
    } else if (errno == EBADF) {
        /*
         * Invalid conversion descriptor.
         * Actually, this shouldn't happen here.
         */
        (void) fprintf(stderr, "Conversion error\\n");
        return (1);
    } else {
        /*
         * This errno is not defined
         */
        (void) fprintf(stderr, "iconv error\\n");
        return (1);
    }
}

/*
 * Initializes the conversion descriptor and outputs
 * the sequence to change the state to initial state.
 */
INIT_SHIFT_STATE(cd, fptr, ileft, tptr, oleft);

(void) iconv_close(cd);
return (0);
}

```

FILES

/usr/lib/iconv/*.so conversion modules
 /usr/lib/iconv/sparcv9/*.so conversion modules

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

iconv(1), **iconv_close(3)**, **iconv_open(3)**, **attributes(5)**, **iconv(5)**, **iconv_unicode(5)**

| NAME | iconv_close – code conversion deallocation function | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | #include <iconv.h> int iconv_close(iconv_t cd); | | | | |
| DESCRIPTION | The iconv_close() function deallocates the conversion descriptor <i>cd</i> and all other associated resources allocated by the iconv_open(3) function. If a file descriptor is used to implement the type <i>iconv_t</i> , that file descriptor will be closed. For examples using the iconv_close() function, see iconv(3) . | | | | |
| RETURN VALUES | Upon successful completion, iconv_close() returns 0; otherwise, it returns -1 and sets <i>errno</i> to indicate the error. | | | | |
| ERRORS | The iconv_close() function may fail if: EBADF The conversion descriptor is invalid. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | iconv(3) , iconv_open(3) , attributes(5) | | | | |

| NAME | iconv_open – code conversion allocation function | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | #include <iconv.h> iconv_t iconv_open(const char * <i>tocode</i> , const char * <i>fromcode</i>); | | | | |
| DESCRIPTION | The iconv_open() function returns a conversion descriptor that describes a conversion from the codeset specified by the string pointed to by the <i>fromcode</i> argument to the codeset specified by the string pointed to by the <i>tocode</i> argument. For state-dependent encodings, the conversion descriptor will be in a codeset-dependent initial shift state, ready for immediate use with the iconv(3) function. Settings of <i>fromcode</i> and <i>tocode</i> and their permitted combinations are implementation-dependent. A conversion descriptor remains valid in a process until that process closes it. For examples using the iconv_open() function, see iconv(3) . | | | | |
| RETURN VALUES | Upon successful completion iconv_open() returns a conversion descriptor for use on subsequent calls to iconv() . Otherwise, iconv_open() returns (<i>iconv_t</i>) -1 and sets <i>errno</i> to indicate the error. | | | | |
| ERRORS | The <i>iconv_open</i> function may fail if: EMFILE { <i>OPEN_MAX</i> } files descriptors are currently open in the calling process. ENFILE Too many files are currently open in the system. ENOMEM Insufficient storage space is available. EINVAL The conversion specified by <i>fromcode</i> and <i>tocode</i> is not supported by the implementation. | | | | |
| ATTRIBUTES | See attributes (5) for descriptions of the following attributes: <table border="1" data-bbox="485 1486 1383 1575"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | iconv(3) , iconv_close(3) , malloc(3C) , attributes(5) | | | | |

NOTES

iconv_open() uses `malloc(3C)` to allocate space for internal buffer areas. **iconv_open()** may fail if there is insufficient storage space to accommodate these buffers.

Portable applications must assume that conversion descriptors are not valid after a call to one of the `exec` functions.

| | |
|----------------------|--|
| NAME | idcok – enable/disable hardware insert-character and delete-character features |
| SYNOPSIS | <pre>#include <curses.h> void idcok(WINDOW *win, bool bf);</pre> |
| PARAMETERS | <p>win Is a pointer to a window.</p> <p>bf Is a Boolean expression.</p> |
| DESCRIPTION | The idcok() function enables or disables the use of hardware insert-character and delete-character features in <i>win</i> . If <i>bf</i> is set to TRUE, the use of these features in <i>win</i> is enabled (if the terminal is equipped). If <i>bf</i> is set to FALSE, their use in <i>win</i> is disabled. |
| RETURN VALUES | The idcok() function does not return a value. |
| ERRORS | None. |
| SEE ALSO | clearok(3XC) , doupdate(3XC) |

NAME | ilogb – returns an unbiased exponent

SYNOPSIS | cc [*flag ...*] *file ...* -lm [*library ...*]
 | #include <math.h>
 | int ilogb(double x);

DESCRIPTION | The **ilogb()** function returns the exponent part of x . Formally, the return value is the integral part of $\log_r |x|$ as a signed integral value, for non-zero finite x , where r is the radix of the machine's floating point arithmetic.

RETURN VALUES | Upon successful completion, **ilogb()** returns the exponent part of x .
 | If x is 0, **ilogb()** returns `-INT_MAX`.
 | If x is NaN or $\pm\text{Inf}$, **ilogb()** returns `INT_MAX`.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **logb(3M)**, **attributes(5)**

| | |
|----------------------|---|
| NAME | immedok – call refresh on changes to window |
| SYNOPSIS | <pre>#include <curses.h> int immedok(WINDOW *win, bool bf);</pre> |
| PARAMETERS | <p>win Is a pointer to the window that is to be refreshed.</p> <p>bf Is a Boolean expression.</p> |
| DESCRIPTION | If <i>bf</i> is TRUE, immedok() calls refresh(3XC) if any change to the window image is made (for example, through functions such as addch(3XC) , clrtoobot(3XC) , and scr1(3XC)). Repeated calls to refresh() may affect performance negatively. The immedok () function is disabled by default. |
| RETRUN VALUES | The immedok() function does not return a value. |
| ERRORS | None. |
| SEE ALSO | addch(3XC) , clearok(3XC) , clrtoobot(3XC) , douupdate(3XC) , scr1(3XC) |

| | |
|----------------------|--|
| NAME | inch, mvinch, mvwinch, winch – return a single-byte character (with rendition) |
| SYNOPSIS | <pre>#include <curses.h> ctype inch(void); ctype mvinch(int y, int x); ctype mvwinch(WINDOW * win, int y, int x); ctype winch(WINDOW * win);</pre> |
| PARAMETERS | <p>y Is the y (row) coordinate of the position of the character to be returned.</p> <p>x Is the x (column) coordinate of the position of the character to be returned.</p> <p>win Is a pointer to the window that contains the character to be returned.</p> |
| DESCRIPTION | <p>The inch() and winch() functions return the <code>ctype</code> character located at the current cursor position of the <code>stdscr</code> window and window <i>win</i> , respectively. The mvinch() and mvwinch() functions return the <code>ctype</code> character located at the position indicated by the <i>x</i> (column) and <i>y</i> (row) parameters (the former in the <code>stdscr</code> window; the latter in window <i>win</i>).</p> <p>The complete character/attribute pair will be returned. The character or attributes can be extracted by performing a bitwise AND on the returned value, using the constants <code>A_CHARTEXT</code> , <code>A_ATTRIBUTES</code> , and <code>A_COLOR</code> .</p> |
| RETURN VALUES | On success, these functions return the specified character and rendition. Otherwise, they return <code>ERR</code> . |
| ERRORS | None. |
| SEE ALSO | addch(3XC) , attroff(3XC) |

| | | | | | | | | | | | |
|---------------------|---|---------------------|---|-----------------|--|-----------------|---|-----------------|--|-------------------|--|
| NAME | inchnstr, inchstr, mvinchnstr, mvinchstr, mvwinchnstr, mvwinchstr, winchnstr, winchstr – retrieve a single-byte character string (with rendition) | | | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int inchnstr(chtype * chstr, int n); int inchstr(chtype * chstr); int mvinchnstr(int y, int x, chtype * chstr, int n); int mvinchstr(int y, int x, chtype * chstr); int mvwinchnstr(WINDOW * win, int y, int x, chtype * chstr, int n); int mvwinchstr(WINDOW * win, int y, int x, chtype * chstr); int winchnstr(WINDOW * win, chtype * chstr, int n); int winchstr(WINDOW * win, chtype * chstr);</pre> | | | | | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="vertical-align: top;"><i>chstr</i></td> <td>Is a pointer to an object that can hold the retrieved character string.</td> </tr> <tr> <td style="vertical-align: top;"><i>n</i></td> <td>Is the number of characters not to exceed when retrieving <i>chstr</i> .</td> </tr> <tr> <td style="vertical-align: top;"><i>y</i></td> <td>Is the y (row) coordinate of the starting position of the string to be retrieved.</td> </tr> <tr> <td style="vertical-align: top;"><i>x</i></td> <td>Is the x (column) coordinate of the starting position of the string to be retrieved.</td> </tr> <tr> <td style="vertical-align: top;"><i>win</i></td> <td>Is a pointer to the window in which the string is to be retrieved.</td> </tr> </table> | <i>chstr</i> | Is a pointer to an object that can hold the retrieved character string. | <i>n</i> | Is the number of characters not to exceed when retrieving <i>chstr</i> . | <i>y</i> | Is the y (row) coordinate of the starting position of the string to be retrieved. | <i>x</i> | Is the x (column) coordinate of the starting position of the string to be retrieved. | <i>win</i> | Is a pointer to the window in which the string is to be retrieved. |
| <i>chstr</i> | Is a pointer to an object that can hold the retrieved character string. | | | | | | | | | | |
| <i>n</i> | Is the number of characters not to exceed when retrieving <i>chstr</i> . | | | | | | | | | | |
| <i>y</i> | Is the y (row) coordinate of the starting position of the string to be retrieved. | | | | | | | | | | |
| <i>x</i> | Is the x (column) coordinate of the starting position of the string to be retrieved. | | | | | | | | | | |
| <i>win</i> | Is a pointer to the window in which the string is to be retrieved. | | | | | | | | | | |
| DESCRIPTION | <p>The inchstr() and winchstr() functions retrieve the character string (with rendition) starting at the current cursor position of the <code>stdscr</code> window and window <i>win</i> , respectively, and ending at the right margin. The mvinchstr() and mvwinchstr() functions retrieve the character string located at the position indicated by the <i>x</i> (column) and <i>y</i> (row) parameters (the former in the <code>stdscr</code> window; the latter in window <i>win</i>).</p> <p>The inchnstr() , winchnstr() , mvinchnstr() , and mvwinchnstr() functions retrieve at most <i>n</i> characters from the window <code>stdscr</code> and <i>win</i> , respectively. The former two functions retrieve the string, starting at the current cursor position; the latter two commands retrieve the string, starting at the position specified by the <i>x</i> and <i>y</i> parameters.</p> | | | | | | | | | | |

All these functions store the retrieved character string in the object pointed to by *chstr* .

The complete character/attribute pair is retrieved. The character or attributes can be extracted by performing a bitwise AND on the retrieved value, using the constants `A_CHARTEXT` , `A_ATTRIBUTES` , and `A_COLOR` . The character string can also be retrieved without attributes by using `instr(3XC)` set of functions.

RETURN VALUES

On success, these functions return `OK` . Otherwise, they return `ERR` .

ERRORS

None.

SEE ALSO

`inch(3XC)` , `innstr(3XC)`

| | |
|--------------------|--|
| NAME | index, rindex – string operations |
| SYNOPSIS | <pre>#include <strings.h> char * index(const char * s, int c); char * rindex(const char * s, int c);</pre> |
| DESCRIPTION | <p>The index() and rindex() functions operate on null-terminated strings.</p> <p>The index() function returns a pointer to the first occurrence of character <i>c</i> in string <i>s</i>.</p> <p>The rindex() function returns a pointer to the last occurrence of character <i>c</i> in string <i>s</i>.</p> <p>Both index() and rindex() return a null pointer if <i>c</i> does not occur in the string. The null character terminating a string is considered to be part of the string.</p> |
| USAGE | <p>On most modern computer systems, you can <i>not</i> use a null pointer to indicate a null string. A null pointer is an error and results in an abort of the program. If you wish to indicate a null string, you must use a pointer that points to an explicit null string. On some machines and with some implementations of the C programming language, a null pointer, if dereferenced, would yield a null string. Though often used, this practice is not always portable. Programmers using a null pointer to represent an empty string should be aware of this portability issue. Even on machines where dereferencing a null pointer does not cause an abort of the program, it does not necessarily yield a null string.</p> |
| SEE ALSO | bstring(3C) , malloc(3C) , string(3C) |

| | |
|---------------------------|--|
| NAME | inet, inet_addr, inet_network, inet_makeaddr, inet_lnaof, inet_netof, inet_ntoa - Internet address manipulation |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lsocket -lnsl [<i>library</i> ...] #include <sys/types.h> #include <sys/socket.h> #include <netinet/in.h> #include <arpa/inet.h> unsigned long inet_addr(const char * cp); unsigned long inet_network(const char * cp); struct in_addr inet_makeaddr(const int net, const int lna); int inet_lnaof(const struct in_addr in); int inet_netof(const struct in_addr in); char * inet_ntoa(const struct in_addr in); </pre> |
| DESCRIPTION | <p>The inet_addr() and inet_network() routines interpret character strings representing numbers expressed in the Internet standard ' . ' notation, returning numbers suitable for use as Internet addresses and Internet network numbers, respectively. The routine inet_makeaddr() takes an Internet network number and a local network address and constructs an Internet address from it. The routines inet_netof() and inet_lnaof() break apart Internet host addresses, returning the network number and local network address part, respectively.</p> <p>The routine inet_ntoa() returns a pointer to a string in the base 256 notation d.d.d.d. See INTERNET ADDRESSES .</p> <p>Internet addresses are returned in network order (bytes ordered from left to right). Network numbers and local address parts are returned as machine format integer values.</p> |
| INTERNET ADDRESSES | Values specified using ' . ' notation take one of the following forms: |

```
a.b.c.d
a.b.c
a.b
a
```

When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address.

When a three part address is specified, the last part is interpreted as a 16-bit quantity and placed in the right most two bytes of the network address. This makes the three part address format convenient for specifying Class B network addresses as `128.net.host`.

When a two part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the right most three bytes of the network address. This makes the two part address format convenient for specifying Class A network addresses as `net.host`.

When only one part is given, the value is stored directly in the network address without any byte rearrangement.

Numbers supplied as *parts* in `' . '` notation may be decimal, octal, or hexadecimal, as specified in the C language. For example, a leading `0x` or `0X` implies hexadecimal; otherwise, a leading `0` implies octal; otherwise, the number is interpreted as decimal.

RETURN VALUES

The value `-1` is returned by `inet_addr()` and `inet_network()` for malformed requests.

The routines `inet_netof()` and `inet_lnaof()` break apart Internet host addresses, returning the network number and local network address part, respectively.

The routine `inet_ntoa()` returns a pointer to a string in the base 256 notation `d.d.d.d` described in `INTERNET ADDRESSES`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

`gethostbyname(3N)`, `getnetbyname(3N)`, `hosts(4)`, `networks(4)`, `attributes(5)`, `inet(5)`

NOTES

The return value from `inet_ntoa()` points to a buffer which is overwritten on each call. This buffer is implemented as thread-specific data in multithreaded applications.

BUGS

The problem of host byte ordering versus network byte ordering is confusing. A simple way to specify Class C network addresses in a manner similar to that for Class B and Class A is needed.

| | |
|--------------------|---|
| NAME | inet_addr, inet_network, inet_makeaddr, inet_lnaof, inet_netof, inet_ntoa – Internet address manipulation |
| SYNOPSIS | <pre>cc [flag ...] file ... -lxnet [library ...] #include <arpa/inet.h> in_addr_t inet_addr(const char * cp); in_addr_t inet_lnaof(struct in_addr in); struct in_addr inet_makeaddr(in_addr_t net, in_addr_t lna); in_addr_t inet_netof(struct in_addr in); in_addr_t inet_network(const char * cp); char * inet_ntoa(struct in_addr in);</pre> |
| DESCRIPTION | <p>The inet_addr() function converts the string pointed to by <i>cp</i>, in the Internet standard dot notation, to an integer value suitable for use as an Internet address.</p> <p>The inet_lnaof() function takes an Internet host address specified by <i>in</i> and extracts the local network address part, in host byte order.</p> <p>The inet_makeaddr() function takes the Internet network number specified by <i>net</i> and the local network address specified by <i>lna</i>, both in host byte order, and constructs an Internet address from them.</p> <p>The inet_netof() function takes an Internet host address specified by <i>in</i> and extracts the network number part, in host byte order.</p> <p>The inet_network() function converts the string pointed to by <i>cp</i>, in the Internet standard dot notation, to an integer value suitable for use as an Internet network number.</p> <p>The inet_ntoa() function converts the Internet host address specified by <i>in</i> to a string in the Internet standard dot notation.</p> |

All Internet addresses are returned in network order (bytes ordered from left to right).

Values specified using dot notation take one of the following forms:

a.b.c.d When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address.

a.b.c When a three-part address is specified, the last part is interpreted as a 16-bit quantity and placed in the rightmost two bytes of the network address. This makes the three-part address format convenient for specifying Class B network addresses as *128.net.host*.

a.b When a two-part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the rightmost three bytes of the network address. This makes the two-part address format convenient for specifying Class A network addresses as *net.host*.

a When only one part is given, the value is stored directly in the network address without any byte rearrangement.

All numbers supplied as parts in dot notation may be decimal, octal, or hexadecimal, that is, a leading 0x or 0X implies hexadecimal, as specified in the *ISO C* standard; otherwise, a leading 0 implies octal; otherwise, the number is interpreted as decimal).

USAGE

The return value of **inet_ntoa()** may point to static data that may be overwritten by subsequent calls to **inet_ntoa()**.

RETURN VALUES

Upon successful completion, **inet_addr()** returns the Internet address. Otherwise, it returns `(in_addr_t)(-1)`.

Upon successful completion, **inet_network()** returns the converted Internet network number. Otherwise, it returns `(in_addr_t)(-1)`.

The **inet_makeaddr()** function returns the constructed Internet address.

The **inet_lnaof()** function returns the local network address part.

The **inet_netof()** function returns the network number.

The **inet_ntoa()** function returns a pointer to the network address in Internet-standard dot notation.

ERRORS

No errors are defined.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `endhostent(3XN)`, `endnetent(3XN)`, `attributes(5)`

| NAME | initgroups - initialize the supplementary group access list | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>#include <grp.h> #include <sys/types.h> int initgroups(const char *name, gid_t basegid);</pre> | | | | |
| DESCRIPTION | <p>The initgroups() function reads the group database to get the group membership for the user specified by <i>name</i>, and initializes the supplementary group access list of the calling process (see getgrnam(3C) and getgroups(2)). The <i>basegid</i> group ID is also included in the supplementary group access list. This is typically the real group ID from the user database.</p> <p>While scanning the group database, if the number of groups, including the <i>basegid</i> entry, exceeds NGROUPS_MAX, subsequent group entries are ignored.</p> | | | | |
| RETURN VALUES | Upon successful completion, 0 is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error. | | | | |
| ERRORS | <p>The initgroups() function will fail and not change the supplementary group access list if:</p> <p>EPERM The effective user ID is not super-user.</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | getgroups(2) , getgrnam(3C) , attributes(5) | | | | |

| | |
|----------------------|--|
| NAME | initscr, newterm – screen initialization functions |
| SYNOPSIS | <pre>#include <curses.h> WINDOW * initscr(void); SCREEN * newterm(char * type, FILE * outfp, FILE * infp);</pre> |
| PARAMETERS | <p><i>type</i> Is a string defining the terminal type to be used in place of TERM .</p> <p><i>outfp</i> Is a pointer to a file to be used for output to the terminal.</p> <p><i>infp</i> Is the pointer to a file to be used for input to the terminal.</p> |
| DESCRIPTION | <p>The initscr() function initializes X/Open Curses data structures, determines the terminal type, and ensures the first call to refresh(3XC) clears the screen.</p> <p>The newterm() function opens a new terminal with each call. It should be used instead of initscr() when the program interacts with more than one terminal. It returns a variable of type <code>SCREEN</code> , which should be used for later reference to that terminal. Before program termination, endwin() should be called for each terminal.</p> <p>The only functions that you can call before calling initscr() or newterm() are filter(3XC) , ripoffline(3XC) , slk_init(3XC) , and use_env(3XC) .</p> |
| RETURN VALUES | <p>On success, the initscr() function returns a pointer to <code>stdscr</code> ; otherwise, initscr() does not return.</p> <p>On success, the newterm() function returns a pointer to the specified terminal; otherwise, a null pointer is returned.</p> |
| ERRORS | None. |
| SEE ALSO | del_curterm(3XC) , delscreen(3XC) , doupdate(3XC) , endwin(3XC) , filter(3XC) , slk_attroff(3XC) , use_env(3XC) |

| | |
|--------------------|--|
| NAME | innstr, instr, mvinnstr, mvinstr, mvwinnstr, mvwinstr, winnstr, winstr – retrieve a multibyte character string (without rendition) |
| SYNOPSIS | <pre>#include <curses.h> int innstr(char * str, int n); int instr(char * str); int mvinnstr(int y, int x, char * str, int n); int mvinstr(int y, int x, char * str); int mvwinnstr(WINDOW * win, int y, int x, char * str, int n); int mvwinstr(WINDOW * win, int y, int x, char * str); int winstr(WINDOW * win, char * str); int winnstr(WINDOW * win, char * str, int n);</pre> |
| PARAMETERS | <p><i>str</i> Is a pointer to an object that can hold the retrieved multibyte character string.</p> <p><i>n</i> Is the number of characters not to exceed when retrieving <i>str</i> .</p> <p><i>y</i> Is the y (row) coordinate of the starting position of the string to be retrieved.</p> <p><i>x</i> Is the x (column) coordinate of the starting position of the string to be retrieved.</p> <p><i>win</i> Is a pointer to the window in which the string is to be retrieved.</p> |
| DESCRIPTION | <p>The instr() and winstr() functions retrieve a multibyte character string (without attributes) starting at the current cursor position of the <code>stdscr</code> window and window <i>win</i> , respectively, and ending at the right margin. The mvinstr() and mvwinstr() functions retrieve a multibyte character string located at the position indicated by the <i>x</i> (column) and <i>y</i> (row) parameters (the former in the <code>stdscr</code> window; the latter in window <i>win</i>).</p> <p>The innstr() , winnstr() , mvinnstr() , and mvwinnstr() functions retrieve at most <i>n</i> characters from the window <code>stdscr</code> and <i>win</i> , respectively. The former two functions retrieve the string starting at the current cursor position; the latter two commands return the string, starting at the position specified by the <i>x</i> and <i>y</i> parameters.</p> |

All these functions store the retrieved string in the object pointed to by *str* . They only store complete multibyte characters. If the area pointed to by *str* is not large enough to hold at least one character, these functions fail.

Only the character portion of the character/rendition pair is returned. To return the complete character/rendition pair, use **winchstr()** .

ERRORS

| | |
|-----|------------------------|
| OK | Successful completion. |
| ERR | An error occurred. |

USAGE

All functions except **winnstr()** may be macros.

SEE ALSO

inch(3XC) , **inchstr(3XC)**

| | | | | | | | | | | | |
|--------------------|---|-------------|---|----------|---|----------|---|----------|--|------------|--|
| NAME | innwstr, inwstr, mvinnwstr, mvinnwstr, mvwinnwstr, mvwinwstr, winnwstr, winwstr – retrieve a wide character string (without rendition) | | | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int innwstr(wchar_t * wstr, int n); int inwstr(wchar_t * wstr); int mvinnwstr(int y, int x, wchar_t * wstr, int n); int mvinnwstr(int y, int x, wchar_t * wstr); int mvwinnwstr(WINDOW* win, int y, int x, wchar_t * wstr, int n); int mvwinwstr(WINDOW* win, int y, int x, wchar_t * wstr); int winnwstr(WINDOW* win, wchar_t * wstr); int winwstr(WINDOW* win, wchar_t * wstr, int n);</pre> | | | | | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="padding-right: 20px;">wstr</td> <td>Is a pointer to an object that can hold the retrieved multibyte character string.</td> </tr> <tr> <td>n</td> <td>Is the number of characters not to exceed when retrieving <i>wstr</i> .</td> </tr> <tr> <td>y</td> <td>Is the y (row) coordinate of the starting position of the string to be retrieved.</td> </tr> <tr> <td>x</td> <td>Is the x (column) coordinate of the starting position of the string to be retrieved.</td> </tr> <tr> <td>win</td> <td>Is a pointer to the window in which the string is to be retrieved.</td> </tr> </table> | wstr | Is a pointer to an object that can hold the retrieved multibyte character string. | n | Is the number of characters not to exceed when retrieving <i>wstr</i> . | y | Is the y (row) coordinate of the starting position of the string to be retrieved. | x | Is the x (column) coordinate of the starting position of the string to be retrieved. | win | Is a pointer to the window in which the string is to be retrieved. |
| wstr | Is a pointer to an object that can hold the retrieved multibyte character string. | | | | | | | | | | |
| n | Is the number of characters not to exceed when retrieving <i>wstr</i> . | | | | | | | | | | |
| y | Is the y (row) coordinate of the starting position of the string to be retrieved. | | | | | | | | | | |
| x | Is the x (column) coordinate of the starting position of the string to be retrieved. | | | | | | | | | | |
| win | Is a pointer to the window in which the string is to be retrieved. | | | | | | | | | | |
| DESCRIPTION | <p>The inwstr() and winwstr() functions retrieve a wide character string (without attributes) starting at the current cursor position of the <code>stdscr</code> window and window <i>win</i> , respectively, and ending at the right margin. The mvinnwstr() and mvwinwstr() functions retrieve a wide character string located at the position indicated by the <i>x</i> (column) and <i>y</i> (row) parameters (the former in the <code>stdscr</code> window; the latter in window <i>win</i>).</p> <p>The innwstr() , winnwstr() , mvinnwstr() , and mvwinnwstr() functions retrieve at most <i>n</i> characters from the window <code>stdscr</code> and <i>win</i> , respectively. The former two functions retrieve the string starting at the current cursor position; the latter two commands return the string, starting at the position specified by the <i>x</i> and <i>y</i> parameters.</p> | | | | | | | | | | |

All these functions store the retrieved string in the object pointed to by *wstr* . They only store complete wide characters. If the area pointed to by *wstr* is not large enough to hold at least one character, these functions fail.

Only the character portion of the character/rendition pair is returned. To return the complete character/rendition pair, use `win_wchstr(3XC)` .

RETURN VALUES

On success, the `inwstr()` , `mvinwstr()` , `mvwinwstr()` , and `winwstr()` functions return `OK` . Otherwise, they return `ERR` .

On success, the `innwstr()` , `mvinnwstr()` , `mvwinnwstr()` , and `winnwstr()` functions return the number of characters read into the string. Otherwise, they return `ERR` .

ERRORS

None.

SEE ALSO

`in_wch(3XC)` , `in_wchnstr(3XC)`

| | | | | | | | | | |
|----------------------|---|------------------|----------------------------------|-----------------|---|-----------------|--|-------------------|--|
| NAME | insch, wunsch, mvinsch, mvwansch – insert a character | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int insch(chtype ch); int mvinsch(int y, int x, chtype ch); int mvwansch(WINDOW * win, int y, int x, chtype ch); int wansch(WINDOW * win, chtype ch);</pre> | | | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="padding-right: 20px;"><i>ch</i></td> <td>Is the character to be inserted.</td> </tr> <tr> <td style="padding-right: 20px;"><i>y</i></td> <td>Is the y (row) coordinate of the position of the character.</td> </tr> <tr> <td style="padding-right: 20px;"><i>x</i></td> <td>Is the x (column) coordinate of the position of the character.</td> </tr> <tr> <td style="padding-right: 20px;"><i>win</i></td> <td>Is a pointer to the window in which the character is to be inserted.</td> </tr> </table> | <i>ch</i> | Is the character to be inserted. | <i>y</i> | Is the y (row) coordinate of the position of the character. | <i>x</i> | Is the x (column) coordinate of the position of the character. | <i>win</i> | Is a pointer to the window in which the character is to be inserted. |
| <i>ch</i> | Is the character to be inserted. | | | | | | | | |
| <i>y</i> | Is the y (row) coordinate of the position of the character. | | | | | | | | |
| <i>x</i> | Is the x (column) coordinate of the position of the character. | | | | | | | | |
| <i>win</i> | Is a pointer to the window in which the character is to be inserted. | | | | | | | | |
| DESCRIPTION | <p>The insch() function inserts the <code>chtype</code> character <i>ch</i> at the current cursor position of the <code>stdscr</code> window. The wansch() function performs the identical action but in window <i>win</i> . The mvinsch() and mvwansch() functions insert the character at the position indicated by the <i>x</i> (column) and <i>y</i> (row) parameters (the former in the <code>stdscr</code> window; the latter in window <i>win</i>). The cursor position does not change.</p> <p>All characters to the right of the inserted character are moved right one character. The last character on the line is deleted.</p> <p>Insertions and deletions occur at the character level. The cursor is adjusted to the first column of the character prior to the the operation.</p> | | | | | | | | |
| RETURN VALUES | On success, these functions return <code>OK</code> . Otherwise, they return <code>ERR</code> . | | | | | | | | |
| ERRORS | None. | | | | | | | | |
| SEE ALSO | <code>delch(3XC)</code> , <code>insnstr(3XC)</code> | | | | | | | | |

| | |
|----------------------|---|
| NAME | insdelln, winsdelln – insert/delete lines to/from the window |
| SYNOPSIS | <pre>#include <curses.h> int insdelln(int n); int winsdelln(WINDOW * win, int n);</pre> |
| PARAMETERS | <p><i>n</i> Is the number of lines to insert or delete (positive <i>n</i> inserts; negative <i>n</i> deletes).</p> <p><i>win</i> Is a pointer to the window in which to insert or delete a line.</p> |
| DESCRIPTION | The insdelln() and winsdelln() functions insert or delete blank lines in <code>stdscr</code> or <i>win</i> , respectively. When <i>n</i> is positive, <i>n</i> lines are added before the current line and the bottom <i>n</i> lines are lost; when <i>n</i> is negative, <i>n</i> lines are deleted starting with the current line, the remaining lines are moved up, and the bottom <i>n</i> lines are cleared. The position of the cursor does not change. |
| RETURN VALUES | On success, these functions return <code>OK</code> . Otherwise, they return <code>ERR</code> . |
| ERRORS | None. |
| SEE ALSO | <code>deleteln(3XC)</code> , <code>insertln(3XC)</code> |

| | |
|----------------------|---|
| NAME | insertln, winsertln – insert a line in a window |
| SYNOPSIS | <pre>#include <curses.h> int insertln(void); int winsertln(WINDOW * win);</pre> |
| PARAMETERS | <p>win Is a pointer to the window in which to insert the line.</p> |
| DESCRIPTION | <p>The insertln() and winsertln() functions insert a blank line before the current line in <code>stdscr</code> or <code>win</code>, respectively. The new line becomes the current line. The current line and all lines after it in the window are moved down one line. The bottom line in the window is discarded.</p> |
| RETURN VALUES | <p>On success, these functions return <code>OK</code>. Otherwise, they return <code>ERR</code>.</p> |
| ERRORS | <p>None.</p> |
| SEE ALSO | <p><code>bkgdset(3XC)</code>, <code>delete1n(3XC)</code>, <code>insdelln(3XC)</code></p> |

| | | | | | | | | | | | |
|--------------------|--|-------------------|--|-----------------|--|-----------------|---|-----------------|--|-------------------|---|
| NAME | insnstr, insstr, mvinsnstr, mvinsstr, mvwinsnstr, mvwinsstr, winsnstr, winsstr – insert a multibyte character string | | | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int insnstr(const char * str, int n); int insstr(const char * str); int mvinsnstr(int y, int x, const char * str, int n); int mvinsstr(int y, int x, const char * str); int mvwinsnstr(WINDOW * win, int y, int x, const char * str, int n); int mvwinsstr(WINDOW * win, int y, int x, const char * str); int winsnstr(WINDOW * win, const char * str, int n); int winsstr(WINDOW * win, const char * str);</pre> | | | | | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="padding-right: 20px;"><i>str</i></td> <td>Is a pointer to the string to be inserted.</td> </tr> <tr> <td><i>n</i></td> <td>Is the number of characters not to exceed when inserting <i>str</i> . If <i>n</i> is less than 1, the entire string is inserted.</td> </tr> <tr> <td><i>y</i></td> <td>Is the y (row) coordinate of the starting position of the string.</td> </tr> <tr> <td><i>x</i></td> <td>Is the x (column) coordinate of the starting position of the string.</td> </tr> <tr> <td><i>win</i></td> <td>Is a pointer to the window in which the string is to be inserted.</td> </tr> </table> | <i>str</i> | Is a pointer to the string to be inserted. | <i>n</i> | Is the number of characters not to exceed when inserting <i>str</i> . If <i>n</i> is less than 1, the entire string is inserted. | <i>y</i> | Is the y (row) coordinate of the starting position of the string. | <i>x</i> | Is the x (column) coordinate of the starting position of the string. | <i>win</i> | Is a pointer to the window in which the string is to be inserted. |
| <i>str</i> | Is a pointer to the string to be inserted. | | | | | | | | | | |
| <i>n</i> | Is the number of characters not to exceed when inserting <i>str</i> . If <i>n</i> is less than 1, the entire string is inserted. | | | | | | | | | | |
| <i>y</i> | Is the y (row) coordinate of the starting position of the string. | | | | | | | | | | |
| <i>x</i> | Is the x (column) coordinate of the starting position of the string. | | | | | | | | | | |
| <i>win</i> | Is a pointer to the window in which the string is to be inserted. | | | | | | | | | | |
| DESCRIPTION | <p>The insstr() function inserts <i>str</i> at the current cursor position of the <code>stdscr</code> window. The winsnstr() function performs the identical action, but in window <i>win</i> . The mvinsstr() and mvwinsstr() functions insert the character string at the starting position indicated by the <i>x</i> (column) and <i>y</i> (row) parameters (the former to the <code>stdscr</code> window; the latter to window <i>win</i>).</p> <p>The insnstr() , winsnstr() , mvinsnstr() , and mvwinsnstr() functions insert <i>n</i> characters to the window or as many as will fit on the line. If <i>n</i> is less than 1, the entire string is inserted or as much of it as fits on the line. The former two functions place the string at the current cursor position; the latter two commands use the position specified by the <i>x</i> and <i>y</i> parameters.</p> | | | | | | | | | | |

All characters to the right of inserted characters are moved to the right. Characters that don't fit on the current line are discarded. The cursor is left at the point of insertion.

If a character in *str* is a newline, carriage return, backspace, or tab, the cursor is moved appropriately. The cursor is moved to the next tab stop for each tab character (by default, tabs are eight characters apart). If the character is a control character other than those previously mentioned, the character is inserted using `^ x` notation, where *x* is a printable character. `clrtoeol(3XC)` is automatically done before a newline.

RETURN VALUES

On success, these functions return `OK`. Otherwise, they return `ERR`.

ERRORS

None.

SEE ALSO

`addchstr(3XC)`, `addstr(3XC)`, `clrtoeol(3XC)`, `ins_nwstr(3XC)`, `insch(3XC)`

| | | | | | | | | | | | |
|--------------------|--|-------------|--|----------|---|----------|---|----------|--|------------|---|
| NAME | ins_nwstr, ins_wstr, mvins_nwstr, mvins_wstr, mvwins_nwstr, mvwins_wstr, wins_nwstr, wins_wstr – insert a wide character string | | | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int ins_nwstr(const wchar_t * wstr, int n); int ins_wstr(const wchar_t * wstr); int mvins_nwstr(int y, int x, const wchar_t * wstr, int n); int mvins_wstr(int y, int x, const wchar_t * wstr); int mvwins_nwstr(WINDOW * win, int y, int x, const wchar_t * wstr, int n); int mvwins_wstr(WINDOW * win, int y, int x, const wchar_t * wstr); int wins_nwstr(WINDOW * win, const wchar_t * wstr, int n); int wins_wstr(WINDOW * win, const wchar_t * wstr);</pre> | | | | | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="padding-right: 20px;">wstr</td> <td>Is a pointer to the string to be inserted.</td> </tr> <tr> <td>n</td> <td>Is the number of characters not to exceed when inserting <i>wstr</i> . If <i>n</i> is less than 1, the entire string is inserted.</td> </tr> <tr> <td>y</td> <td>Is the y (row) coordinate of the starting position of the string.</td> </tr> <tr> <td>x</td> <td>Is the x (column) coordinate of the starting position of the string.</td> </tr> <tr> <td>win</td> <td>Is a pointer to the window in which the string is to be inserted.</td> </tr> </table> | wstr | Is a pointer to the string to be inserted. | n | Is the number of characters not to exceed when inserting <i>wstr</i> . If <i>n</i> is less than 1, the entire string is inserted. | y | Is the y (row) coordinate of the starting position of the string. | x | Is the x (column) coordinate of the starting position of the string. | win | Is a pointer to the window in which the string is to be inserted. |
| wstr | Is a pointer to the string to be inserted. | | | | | | | | | | |
| n | Is the number of characters not to exceed when inserting <i>wstr</i> . If <i>n</i> is less than 1, the entire string is inserted. | | | | | | | | | | |
| y | Is the y (row) coordinate of the starting position of the string. | | | | | | | | | | |
| x | Is the x (column) coordinate of the starting position of the string. | | | | | | | | | | |
| win | Is a pointer to the window in which the string is to be inserted. | | | | | | | | | | |
| DESCRIPTION | <p>The ins_wstr() function inserts <i>wstr</i> at the current cursor position of the <code>stdscr</code> window. The wins_wstr() function performs the identical action, but in window <i>win</i> . The mvins_wstr() and mvwins_wstr() functions insert <i>wstr</i> string at the starting position indicated by the <i>x</i> (column) and <i>y</i> (row) parameters (the former in the <code>stdscr</code> window; the latter in window <i>win</i>).</p> <p>The ins_nwstr() , wins_nwstr() , mvins_nwstr() , and mvwins_nwstr() functions insert <i>n</i> characters to the window or as many as will fit on the line. If <i>n</i> is less than 1, the entire string is inserted or as much of it as fits on the line. The former two functions place the string at the current cursor position; the latter two commands use the position specified by the <i>x</i> and <i>y</i> parameters.</p> | | | | | | | | | | |

All characters to the right of inserted characters are moved to the right. Characters that don't fit on the current line are discarded. The cursor is left at the point of insertion.

If a character in *wstr* is a newline, carriage return, backspace, or tab, the cursor is moved appropriately. The cursor is moved to the next tab stop for each tab character (by default, tabs are eight characters apart). If the character is a control character other than those previously mentioned, the character is inserted using $\wedge x$ notation, where *x* is a printable character. `clrtoeol(3XC)` is automatically done before a newline.

RETURN VALUES

On success, these functions return `OK`. Otherwise, they return `ERR`.

ERRORS

None.

SEE ALSO

`add_wchnstr(3XC)`, `addnwstr(3XC)`, `clrtoeol(3XC)`, `ins_wch(3XC)`, `insnstr(3XC)`

NAME | insque, remque – insert/remove element from a queue

SYNOPSIS | include <search.h>

| void **insque**(struct qelem * *elem*, struct qelem * *pred*);

| void **remque**(struct qelem * *elem*);

DESCRIPTION | The **insque()** and **remque()** functions manipulate queues built from doubly linked lists. Each element in the queue must be in the following form:

```

struct qelem {
    struct qelem *q_forw;
    struct qelem *q_back;
    char
q_data[ ];
};

```

| The **insque()** function inserts *elem* in a queue immediately after *pred* . The **remque()** function removes an entry *elem* from a queue.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO | **attributes(5)**

| | | | | | | | | | |
|----------------------|--|------------|--|----------|---|----------|--|------------|--|
| NAME | ins_wch, wins_wch, mvins_wch, mvwins_wch – insert a complex character | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int ins_wch(const cchar_t * wch); int mvins_wch(int y, int x, const cchar_t * wch); int mvwins_wch(WINDOW * win, int y, int x, const cchar_t * wch); int wins_wch(WINDOW * win, const cchar_t * wch);</pre> | | | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="padding-right: 20px;">wch</td> <td>Is the complex character to be inserted.</td> </tr> <tr> <td>y</td> <td>Is the y (row) coordinate of the position of the character.</td> </tr> <tr> <td>x</td> <td>Is the x (column) coordinate of the position of the character.</td> </tr> <tr> <td>win</td> <td>Is a pointer to the window in which the character is to be inserted.</td> </tr> </table> | wch | Is the complex character to be inserted. | y | Is the y (row) coordinate of the position of the character. | x | Is the x (column) coordinate of the position of the character. | win | Is a pointer to the window in which the character is to be inserted. |
| wch | Is the complex character to be inserted. | | | | | | | | |
| y | Is the y (row) coordinate of the position of the character. | | | | | | | | |
| x | Is the x (column) coordinate of the position of the character. | | | | | | | | |
| win | Is a pointer to the window in which the character is to be inserted. | | | | | | | | |
| DESCRIPTION | <p>The ins_wch() function inserts the complex character <i>wch</i> at the current cursor position of the <code>stdscr</code> window. The wins_wch() function performs the identical action but in window <i>win</i>. The mvins_wch() and mvwins_wch() functions insert the character at the position indicated by the <i>x</i> (column) and <i>y</i> (row) parameters (the former in the <code>stdscr</code> window; the latter in window <i>win</i>). The cursor position does not change.</p> <p>All characters to the right of the inserted character are moved right one character. The last character on the line is deleted.</p> <p>Insertions and deletions occur at the character level. The cursor is adjusted to the first column of the character prior to the the operation.</p> | | | | | | | | |
| RETURN VALUES | On success, these functions return <code>OK</code> . Otherwise, they return <code>ERR</code> . | | | | | | | | |
| ERRORS | None. | | | | | | | | |
| SEE ALSO | <code>add_wch(3XC)</code> , <code>ins_nwstr(3XC)</code> | | | | | | | | |

| | |
|----------------------|---|
| NAME | intrflush – flush output in tty on interrupt |
| SYNOPSIS | <pre>#include <curses.h> int intrflush(WINDOW *win, bool bf);</pre> |
| PARAMETERS | <p>win Is ignored.</p> <p>bf Is a Boolean expression.</p> |
| DESCRIPTION | <p>If this option is enabled (<i>bf</i> is TRUE), intrflush() flushes all output in the terminal driver when an interrupt, quit, or suspend character is sent to the terminal. This increases interrupt response time but causes X/Open Curses to lose track of what currently exists on the screen. If this option is disabled (<i>bf</i> is FALSE), intrflush() does not flush output on an interrupt, quit, or suspend character. Whether this option is enabled or disabled by default depends on the tty driver.</p> |
| RETURN VALUES | On success, these functions return OK. Otherwise, they return ERR. |
| ERRORS | None. |
| SEE ALSO | flushinp(3XC) , qiflush(3XC) |

| | | | | | | | | | |
|----------------------|---|------------|---|----------|--|----------|---|------------|--|
| NAME | in_wch, mvin_wch, mvwin_wch, win_wch – retrieve a complex character (with rendition) | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int in_wch(cchar_t * wcv); int mvin_wch(int y, int x, cchar_t * wcv); int mvwin_wch(WINDOW * win, int y, cchar_t * wcv); int win_wch(WINDOW * win, cchar_t * wcv);</pre> | | | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="vertical-align: top;">wcv</td> <td>Is a pointer to an object that can store a complex character and its rendition.</td> </tr> <tr> <td style="vertical-align: top;">y</td> <td>Is the y (row) coordinate of the position of the character to be returned.</td> </tr> <tr> <td style="vertical-align: top;">x</td> <td>Is the x (column) coordinate of the position of the character to be returned.</td> </tr> <tr> <td style="vertical-align: top;">win</td> <td>Is a pointer to the window that contains the character to be returned.</td> </tr> </table> | wcv | Is a pointer to an object that can store a complex character and its rendition. | y | Is the y (row) coordinate of the position of the character to be returned. | x | Is the x (column) coordinate of the position of the character to be returned. | win | Is a pointer to the window that contains the character to be returned. |
| wcv | Is a pointer to an object that can store a complex character and its rendition. | | | | | | | | |
| y | Is the y (row) coordinate of the position of the character to be returned. | | | | | | | | |
| x | Is the x (column) coordinate of the position of the character to be returned. | | | | | | | | |
| win | Is a pointer to the window that contains the character to be returned. | | | | | | | | |
| DESCRIPTION | <p>The in_wch() and win_wch() functions retrieve the complex character and its rendition located at the current cursor position of the <code>stdscr</code> window and window <code>win</code>, respectively. The mvin_wch() and mvwin_wch() functions retrieve the complex character and its rendition located at the position indicated by the <code>x</code> (column) and <code>y</code> (row) parameters (the former in the <code>stdscr</code> window; the latter in window <code>win</code>).</p> <p>All these functions store the retrieved character and its rendition in the object pointed to by <code>wcv</code>.</p> | | | | | | | | |
| RETURN VALUES | On success, these functions return <code>OK</code> . Otherwise, they return <code>ERR</code> . | | | | | | | | |
| ERRORS | None. | | | | | | | | |
| SEE ALSO | <code>add_wch(3XC)</code> , <code>inch(3XC)</code> | | | | | | | | |

| | | | | | | | | | | | |
|--------------------|---|---------------|---|----------|---|----------|---|----------|--|------------|--|
| NAME | in_wchnstr, in_wchstr, mvin_wchnstr, mvin_wchstr, mvwin_wchnstr, mvwin_wchstr, win_wchnstr, win_wchstr – retrieve complex character string (with rendition) | | | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int in_wchnstr(cchar_t * wchstr, int n); int in_wchstr(cchar_t * wchstr); int mvin_wchnstr(int y, int x, cchar_t * wchstr, int n); int mvin_wchstr(int y, int x, cchar_t * wchstr); int mvwin_wchnstr(WINDOW * win, int y, int x, cchar_t * wchstr, int n); int mvwin_wchstr(WINDOW * win, int y, int x, cchar_t * wchstr); int win_wchnstr(WINDOW * win, cchar_t * wchstr, int n); int win_wchstr(WINDOW * win, cchar_t * wchstr);</pre> | | | | | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="padding-right: 20px;">wchstr</td> <td>Is a pointer to an object where the retrieved complex character string can be stored.</td> </tr> <tr> <td>n</td> <td>Is the number of characters not to exceed when retrieving <i>wchstr</i> .</td> </tr> <tr> <td>y</td> <td>Is the y (row) coordinate of the starting position of the string to be retrieved.</td> </tr> <tr> <td>x</td> <td>Is the x (column) coordinate of the starting position of the string to be retrieved.</td> </tr> <tr> <td>win</td> <td>Is a pointer to the window in which the string is to be retrieved.</td> </tr> </table> | wchstr | Is a pointer to an object where the retrieved complex character string can be stored. | n | Is the number of characters not to exceed when retrieving <i>wchstr</i> . | y | Is the y (row) coordinate of the starting position of the string to be retrieved. | x | Is the x (column) coordinate of the starting position of the string to be retrieved. | win | Is a pointer to the window in which the string is to be retrieved. |
| wchstr | Is a pointer to an object where the retrieved complex character string can be stored. | | | | | | | | | | |
| n | Is the number of characters not to exceed when retrieving <i>wchstr</i> . | | | | | | | | | | |
| y | Is the y (row) coordinate of the starting position of the string to be retrieved. | | | | | | | | | | |
| x | Is the x (column) coordinate of the starting position of the string to be retrieved. | | | | | | | | | | |
| win | Is a pointer to the window in which the string is to be retrieved. | | | | | | | | | | |
| DESCRIPTION | <p>The in_wchstr() and win_wchstr() functions retrieve a complex character string (with rendition) starting at the current cursor position of the <code>stdscr</code> window and window <i>win</i> , respectively, and ending at the right margin. The mvin_wchstr() and mvwin_wchstr() functions retrieve a complex character string located at the position indicated by the <i>x</i> (column) and <i>y</i> (row) parameters (the former in the <code>stdscr</code> window; the latter in window <i>win</i>).</p> <p>The in_wchnstr() , win_wchnstr() , mvin_wchnstr() , and mvwin_wchnstr() functions retrieve at most <i>n</i> characters from the window <code>stdscr</code> and <i>win</i> , respectively. The former two functions retrieve the string, starting at the</p> | | | | | | | | | | |

current cursor position; the latter two commands retrieve the string, starting at the position specified by the *x* and *y* parameters.

The retrieved character string (with renditions) is stored in the object pointed to by *wcval* .

RETURN VALUES

On success, these functions return *OK* . Otherwise, they return *ERR* .

ERRORS

None.

SEE ALSO

in_wch(3XC)

| | |
|----------------------|---|
| NAME | isaexec – invoke isa-specific executable |
| SYNOPSIS | <pre>#include <unistd.h> int isaexec(const char *path, char *const argv[], char *const envp[]);</pre> |
| DESCRIPTION | <p>The isaexec() function takes the path specified as <i>path</i> and breaks it into directory and file name components. It enquires from the running system the list of supported instruction set architectures; see isalist(5). The function traverses the list for an executable file in named subdirectories of the original directory. When such a file is located, execve() is invoked with <i>argv[]</i> and <i>envp[]</i>. See exec(2).</p> |
| RETURN VALUES | <p>If no file is located, isaexec() returns ENOENT. Other return values are the same as for execve().</p> |
| EXAMPLES | <p>EXAMPLE 1 Example of isaexec() function.</p> <p>On a system whose <i>isalist</i> is</p> <pre>sparcv7 sparc</pre> <p>the program</p> <pre>int main(int argc, char *argv[], char *envp[]) { return (isaexec("/bin/thing", argv, envp)); }</pre> <p>will look first for an executable file named <i>/bin/sparcv7/thing</i>, then for an executable file named <i>bin/sparc/thing</i>. It will invoke execve() on the first executable file it finds named <i>thing</i>.</p> <p>On that same system, a program called <i>/u/bin/tofu</i> can cause either <i>/u/bin/sparcv7/tofu</i> or <i>/u/bin/sparc/tofu</i> to be invoked using the following code:</p> <pre>int main(int argc, char *argv[], char *envp[]) { return (isaexec(getexecname(), argv, envp)); }</pre> |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---------------------|-----------------|
| MT-Level | Safe |
| Interface Stability | Stable |

SEE ALSO

`exec(2)`, `getexecname(3C)`, `attributes(5)`, `isalist(5)`

| NAME | isastream – test a file descriptor | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | #include <stropts.h> int isastream (int <i>fildes</i>); | | | | |
| DESCRIPTION | The isastream() function determines if a file descriptor represents a STREAMS file. The <i>fildes</i> argument refers to an open file descriptor. | | | | |
| RETURN VALUES | Upon successful completion, isastream() returns 1 if <i>fildes</i> represents a STREAMS file, and 0 if it does not. Otherwise, -1 is return and <i>errno</i> is set to indicate the error. | | | | |
| ERRORS | The isastream() function will fail if: EBADF The <i>fildes</i> argument is not a valid file descriptor. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | attributes(5) , streamio(7I) <i>STREAMS Programming Guide</i> | | | | |

| NAME | isatty – test for a terminal device | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | #include <unistd.h> int isatty (int <i>fildev</i>); | | | | |
| DESCRIPTION | The isatty() function tests whether <i>fildev</i> , an open file descriptor, is associated with a terminal device. | | | | |
| RETURN VALUES | The isatty() function returns 1 if <i>fildev</i> is associated with a terminal; otherwise it returns 0 and may set <code>errno</code> to indicate the error. | | | | |
| ERRORS | The isatty() function may fail if: EBADF The <i>fildev</i> argument is not a valid open file descriptor. ENOTTY The <i>fildev</i> argument is not associated with a terminal. | | | | |
| USAGE | The isatty() function does not necessarily indicate that a human being is available for interaction via <i>fildev</i> . It is quite possible that non-terminal devices are connected to the communications line. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | ttyname(3C) , attributes(5) | | | | |

| NAME | isencrypt – determine whether a buffer of characters is encrypted | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag...] [file...] -lgen [library...] #include<libgen.h> int isencrypt(const char *buf, size_t ninbuf);</pre> | | | | |
| DESCRIPTION | <p>isencrypt() uses heuristics to determine whether a buffer of characters is encrypted. It requires two arguments: a pointer to an array of characters and the number of characters in the buffer.</p> <p>isencrypt() assumes that the file is not encrypted if all the characters in the first block are ASCII characters. If there are non-ASCII characters in the first <i>ninbuf</i> characters, and if the setlocale() LC_CTYPE category is set to <code>C</code> or <code>ascii</code>, isencrypt() assumes that the buffer is encrypted</p> <p>If the LC_CTYPE category is set to a value other than <code>C</code> or <code>ascii</code>, then isencrypt() uses a combination of heuristics to determine if the buffer is encrypted. If <i>ninbuf</i> has at least 64 characters, a chi-square test is used to determine if the bytes in the buffer have a uniform distribution; if it does, then isencrypt() assumes the buffer is encrypted. If the buffer has less than 64 characters, a check is made for null characters and a terminating new-line to determine whether the buffer is encrypted.</p> | | | | |
| RETURN VALUES | If the buffer is encrypted, 1 is returned; otherwise, zero is returned. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | setlocale(3C) , attributes(5) | | | | |
| NOTES | When compiling multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the compile line. This flag should only be used in multithreaded applications. | | | | |

| | | | | | | | | | | | | | | | |
|-----------------------|---|-------------------|---|--------------------|---|---------------------|--|---------------------|--|-----------------|---|-----------------|--|-----------------------|---|
| NAME | is_linetouched, is_wintouched, touchline, touchwin, untouchwin, wtouchln – control window refresh | | | | | | | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> bool is_linetouched(WINDOW * win, int line); bool is_wintouchwin(WINDOW * win); int touchline(WINDOW * win, int start, int count); int touchwin(WINDOW * win); int untouchwin(WINDOW * win); int wtouchln(WINDOW * win, int y, int n, int changed);</pre> | | | | | | | | | | | | | | |
| PARAMETERS | <table border="0" style="width: 100%;"> <tr> <td style="vertical-align: top; padding-right: 10px;"><i>win</i></td> <td>Is a pointer to the window in which the refresh is to be controlled or monitored.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 10px;"><i>line</i></td> <td>Is the line to be checked for change since refresh.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 10px;"><i>start</i></td> <td>Is the starting line number of the portion of the window to make appear changed.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 10px;"><i>count</i></td> <td>Is the number of lines in the window to mark as changed.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 10px;"><i>y</i></td> <td>Is the starting line number of the portion of the window to make appear changed or not changed.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 10px;"><i>n</i></td> <td>Is the number of lines in the window to mark as changed.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 10px;"><i>changed</i></td> <td>Is a flag indicating whether to make lines look changed (0) or not changed (1).</td> </tr> </table> | <i>win</i> | Is a pointer to the window in which the refresh is to be controlled or monitored. | <i>line</i> | Is the line to be checked for change since refresh. | <i>start</i> | Is the starting line number of the portion of the window to make appear changed. | <i>count</i> | Is the number of lines in the window to mark as changed. | <i>y</i> | Is the starting line number of the portion of the window to make appear changed or not changed. | <i>n</i> | Is the number of lines in the window to mark as changed. | <i>changed</i> | Is a flag indicating whether to make lines look changed (0) or not changed (1). |
| <i>win</i> | Is a pointer to the window in which the refresh is to be controlled or monitored. | | | | | | | | | | | | | | |
| <i>line</i> | Is the line to be checked for change since refresh. | | | | | | | | | | | | | | |
| <i>start</i> | Is the starting line number of the portion of the window to make appear changed. | | | | | | | | | | | | | | |
| <i>count</i> | Is the number of lines in the window to mark as changed. | | | | | | | | | | | | | | |
| <i>y</i> | Is the starting line number of the portion of the window to make appear changed or not changed. | | | | | | | | | | | | | | |
| <i>n</i> | Is the number of lines in the window to mark as changed. | | | | | | | | | | | | | | |
| <i>changed</i> | Is a flag indicating whether to make lines look changed (0) or not changed (1). | | | | | | | | | | | | | | |
| DESCRIPTION | <p>The touchwin() function marks the entire window as dirty. This makes it appear to X/Open Curses as if the whole window has been changed, thus causing the entire window to be rewritten with the next call to refresh(3XC). This is sometimes necessary when using overlapping windows; the change to one window will not be reflected in the other and, hence will not be recorded.</p> <p>The touchline() function marks as dirty a portion of the window starting at line <i>start</i> and continuing for <i>count</i> lines instead of the entire window. Consequently, that portion of the window is updated with the next call to refresh().</p> | | | | | | | | | | | | | | |

The **untouchwin()** function marks all lines in the window as unchanged since the last refresh, ensuring that it is not updated.

The **wtouchln()** function marks *n* lines starting at line *y* as either changed (*changed* =1) or unchanged (*changed* =0) since the last refresh.

To find out which lines or windows have been changed since the last refresh, use the **is_linetouched()** and **is_wintouched()** commands, respectively. These return **TRUE** if the specified line or window have been changed since the last call to **refresh()** or **FALSE** if no changes have been made.

RETURN VALUES

On success, these functions return **OK** . Otherwise, they return **ERR** .

ERRORS

None.

SEE ALSO

douupdate(3XC)

| | | | | | | | | | | | | | | | | | | | | | |
|----------------------|---|---------|---------------|---------|-----------|---------|-------------------|---------|-------------------|------------|--------------------------------|------------|--------------------------------|----------|---------------|----------|---------------|----------|------------------------------|----------|------------------------------|
| NAME | isnan, isnand, isnanf, finite, fpclass, unordered – determine type of floating-point number | | | | | | | | | | | | | | | | | | | | |
| SYNOPSIS | <pre>#include <ieeefp.h> int isnand(double dsrc); int isnanf(float fsrc); int finite(double dsrc); fpclass_t fpclass(double dsrc); int unordered(double dsrc1, double dsrc2); #include <math.h> int isnan(double dsrc);</pre> | | | | | | | | | | | | | | | | | | | | |
| DESCRIPTION | <p>The isnan() function is identical to the isnand() function.</p> <p>The isnanf() function is implemented as a macro included in the <code><ieeefp.h></code> header.</p> <p>The fpclass() function returns one of the following classes to which <i>dsrc</i> belongs:</p> <table border="0"> <tr> <td>FP_SNAN</td> <td>signaling NaN</td> </tr> <tr> <td>FP_QNAN</td> <td>quiet NaN</td> </tr> <tr> <td>FP_NINF</td> <td>negative infinity</td> </tr> <tr> <td>FP_PINF</td> <td>positive infinity</td> </tr> <tr> <td>FP_NDENORM</td> <td>negative denormalized non-zero</td> </tr> <tr> <td>FP_PDENORM</td> <td>positive denormalized non-zero</td> </tr> <tr> <td>FP_NZERO</td> <td>negative zero</td> </tr> <tr> <td>FP_PZERO</td> <td>positive zero</td> </tr> <tr> <td>FP_NNORM</td> <td>negative normalized non-zero</td> </tr> <tr> <td>FP_PNORM</td> <td>positive normalized non-zero</td> </tr> </table> <p>None of these routines generates an exception, even for signaling NaNs.</p> | FP_SNAN | signaling NaN | FP_QNAN | quiet NaN | FP_NINF | negative infinity | FP_PINF | positive infinity | FP_NDENORM | negative denormalized non-zero | FP_PDENORM | positive denormalized non-zero | FP_NZERO | negative zero | FP_PZERO | positive zero | FP_NNORM | negative normalized non-zero | FP_PNORM | positive normalized non-zero |
| FP_SNAN | signaling NaN | | | | | | | | | | | | | | | | | | | | |
| FP_QNAN | quiet NaN | | | | | | | | | | | | | | | | | | | | |
| FP_NINF | negative infinity | | | | | | | | | | | | | | | | | | | | |
| FP_PINF | positive infinity | | | | | | | | | | | | | | | | | | | | |
| FP_NDENORM | negative denormalized non-zero | | | | | | | | | | | | | | | | | | | | |
| FP_PDENORM | positive denormalized non-zero | | | | | | | | | | | | | | | | | | | | |
| FP_NZERO | negative zero | | | | | | | | | | | | | | | | | | | | |
| FP_PZERO | positive zero | | | | | | | | | | | | | | | | | | | | |
| FP_NNORM | negative normalized non-zero | | | | | | | | | | | | | | | | | | | | |
| FP_PNORM | positive normalized non-zero | | | | | | | | | | | | | | | | | | | | |
| RETURN VALUES | The isnan() , isnand() , and isnanf() function return <code>TRUE (1)</code> if the argument <i>dsrc</i> or <i>fsrc</i> is a NaN; otherwise they return <code>FALSE (0)</code> . | | | | | | | | | | | | | | | | | | | | |

The **finite()** function returns `TRUE (1)` if the argument *dsrc* is neither infinity nor NaN; otherwise it returns `FALSE (0)`.

The **unordered()** function returns `TRUE (1)` if one of its two arguments is unordered with respect to the other argument. This is equivalent to reporting whether either argument is NaN. If neither argument is NaN, `FALSE (0)` is returned.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

fpgetround(3C) , **attributes(5)**

NAME | isnan – test for NaN

SYNOPSIS | cc [*flag ...*] *file ...* -lm [*library ...*]
 | #include <math.h>
 | int **isnan**(double *x*);

DESCRIPTION | The **isnan()** function tests whether *x* is NaN.

RETURN VALUES | The **isnan()** function returns non-zero if *x* is NaN. Otherwise, 0 is returned.

USAGE | On systems not supporting NaN, **isnan()** always returns 0.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **attributes(5)**

| | |
|--------------------|--|
| NAME | iswalpha, iswupper, iswlower, iswdigit, iswxdigit, iswalnum, iswspace, iswpunct, iswprint, iswcntrl, iswascii, iswgraph, isphonogram, isideogram, isenglish, isnumber, isspecial – wide-character code classification functions |
| SYNOPSIS | <pre>#include <wchar.h> int iswalpha(wint_t wc);</pre> |
| DESCRIPTION | <p>These functions test whether <i>wc</i> is a wide-character code representing a character of a particular class defined in the LC_CTYPE category of the current locale.</p> <p>In all cases, <i>wc</i> is a <code>wint_t</code>, the value of which must be a wide-character code corresponding to a valid character in the current locale or must equal the value of the macro <code>WEOF</code>. If the argument has any other values, the behavior is undefined.</p> <p><code>iswalpha(<i>wc</i>)</code> Tests whether <i>wc</i> is a wide-character code representing a character of class "alpha" in the program's current locale.</p> <p><code>iswupper(<i>wc</i>)</code> Tests whether <i>wc</i> is a wide-character code representing a character of class "upper" in the program's current locale.</p> <p><code>iswlower(<i>wc</i>)</code> Tests whether <i>wc</i> is a wide-character code representing a character of class "lower" in the program's current locale.</p> <p><code>iswdigit(<i>wc</i>)</code> Tests whether <i>wc</i> is a wide-character code representing a character of class "digit" in the program's current locale.</p> <p><code>iswxdigit(<i>wc</i>)</code> Tests whether <i>wc</i> is a wide-character code representing a character of class "xdigit" in the program's current locale.</p> <p><code>iswalnum(<i>wc</i>)</code> Tests whether <i>wc</i> is a wide-character code representing a character of class "alpha" or "digit" in the program's current locale.</p> <p><code>iswspace(<i>wc</i>)</code> Tests whether <i>wc</i> is a wide-character code representing a character of class "space" in the program's current locale.</p> <p><code>iswpunct(<i>wc</i>)</code> Tests whether <i>wc</i> is a wide-character code representing a character of class "punct" in the program's current locale.</p> |

| | |
|--------------------------------|--|
| <code>iswprint(wc)</code> | Tests whether <code>wc</code> is a wide-character code representing a character of class "print" in the program's current locale. |
| <code>iswgraph(wc)</code> | Tests whether <code>wc</code> is a wide-character code representing a character of class "graph" in the program's current locale. |
| <code>iswcntrl(wc)</code> | Tests whether <code>wc</code> is a wide-character code representing a character of class "cntrl" in the program's current locale. |
| <code>iswascii(wc)</code> | Tests whether <code>wc</code> is a wide-character code representing an ASCII character. |
| <code>isphonogram(wc)</code> | Tests whether <code>wc</code> is a wide-character code representing a phonetic language character, excluding ASCII characters. |
| <code>isideogram(wc)</code> | Tests whether <code>wc</code> is a wide-character code representing an ideographic language character, excluding ASCII characters. |
| <code>isenglish(wc)</code> | Tests whether <code>wc</code> is a wide-character code representing an English language character, excluding ASCII characters. |
| <code>isnumber(wc)</code> | Tests whether <code>wc</code> is a wide-character code representing digit [0–9], excluding ASCII characters. |
| <code>isspecial(wc)</code> | Tests whether <code>wc</code> is a wide-character code representing a special language character, excluding ASCII characters. |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |
| CSI | Enabled |

SEE ALSO

`localedef(1)`, `setlocale(3C)`, `stdio(3S)`, `ascii(5)`, `attributes(5)`

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|--|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|----------|-----------------------|--|-----------------------|--|-----------------------|--|-----------------------|--|-----------------------|--|-----------------------|--|-----------------------|--|-----------------------|--|-----------------------|--|-----------------------|--|------------------------|---|
| NAME | iswctype – test character for specified class | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SYNOPSIS | <pre>#include <wchar.h> int iswctype(wint_t wc, wctype_t charclass);</pre> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| DESCRIPTION | <p>The iswctype() function determines whether the wide-character code <i>wc</i> has the character class <i>charclass</i>, returning TRUE or FALSE. The iswctype() function is defined on WEOF and wide-character codes corresponding to the valid character encodings in the current locale. If the <i>wc</i> argument is not in the domain of the function, the result is undefined. If the value of <i>charclass</i> is invalid (that is, not obtained by a call to wctype(3C) or <i>charclass</i> is invalidated by a subsequent call to setlocale(3C) that has affected category LC_CTYPE), the result is indeterminate.</p> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RETURN VALUES | The iswctype() function returns 0 for FALSE and non-zero for TRUE . | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| USAGE | <p>There are twelve strings that are reserved for the standard character classes:</p> <table border="1" data-bbox="485 911 1383 1085"> <tbody> <tr> <td>"alnum"</td> <td>"alpha"</td> <td>"blank"</td> </tr> <tr> <td>"cntrl"</td> <td>"digit"</td> <td>"graph"</td> </tr> <tr> <td>"lower"</td> <td>"print"</td> <td>"punct"</td> </tr> <tr> <td>"space"</td> <td>"upper"</td> <td>"xdigit"</td> </tr> </tbody> </table> <p>In the table below, the functions in the left column are equivalent to the functions in the right column.</p> <table border="1" data-bbox="485 1203 1383 1682"> <tbody> <tr> <td>iswalnum(<i>wc</i>)</td> <td>iswctype(<i>wc</i>, wctype("alnum"))</td> </tr> <tr> <td>iswalpha(<i>wc</i>)</td> <td>iswctype(<i>wc</i>, wctype("alpha"))</td> </tr> <tr> <td>iswcntrl(<i>wc</i>)</td> <td>iswctype(<i>wc</i>, wctype("cntrl"))</td> </tr> <tr> <td>iswdigit(<i>wc</i>)</td> <td>iswctype(<i>wc</i>, wctype("digit"))</td> </tr> <tr> <td>iswgraph(<i>wc</i>)</td> <td>iswctype(<i>wc</i>, wctype("graph"))</td> </tr> <tr> <td>iswlower(<i>wc</i>)</td> <td>iswctype(<i>wc</i>, wctype("lower"))</td> </tr> <tr> <td>iswprint(<i>wc</i>)</td> <td>iswctype(<i>wc</i>, wctype("print"))</td> </tr> <tr> <td>iswpunct(<i>wc</i>)</td> <td>iswctype(<i>wc</i>, wctype("punct"))</td> </tr> <tr> <td>iswspace(<i>wc</i>)</td> <td>iswctype(<i>wc</i>, wctype("space"))</td> </tr> <tr> <td>iswupper(<i>wc</i>)</td> <td>iswctype(<i>wc</i>, wctype("upper"))</td> </tr> <tr> <td>iswxdigit(<i>wc</i>)</td> <td>iswctype(<i>wc</i>, wctype("xdigit"))</td> </tr> </tbody> </table> <p>The call</p> | "alnum" | "alpha" | "blank" | "cntrl" | "digit" | "graph" | "lower" | "print" | "punct" | "space" | "upper" | "xdigit" | iswalnum(<i>wc</i>) | iswctype(<i>wc</i> , wctype("alnum")) | iswalpha(<i>wc</i>) | iswctype(<i>wc</i> , wctype("alpha")) | iswcntrl(<i>wc</i>) | iswctype(<i>wc</i> , wctype("cntrl")) | iswdigit(<i>wc</i>) | iswctype(<i>wc</i> , wctype("digit")) | iswgraph(<i>wc</i>) | iswctype(<i>wc</i> , wctype("graph")) | iswlower(<i>wc</i>) | iswctype(<i>wc</i> , wctype("lower")) | iswprint(<i>wc</i>) | iswctype(<i>wc</i> , wctype("print")) | iswpunct(<i>wc</i>) | iswctype(<i>wc</i> , wctype("punct")) | iswspace(<i>wc</i>) | iswctype(<i>wc</i> , wctype("space")) | iswupper(<i>wc</i>) | iswctype(<i>wc</i> , wctype("upper")) | iswxdigit(<i>wc</i>) | iswctype(<i>wc</i> , wctype("xdigit")) |
| "alnum" | "alpha" | "blank" | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| "cntrl" | "digit" | "graph" | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| "lower" | "print" | "punct" | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| "space" | "upper" | "xdigit" | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| iswalnum(<i>wc</i>) | iswctype(<i>wc</i> , wctype("alnum")) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| iswalpha(<i>wc</i>) | iswctype(<i>wc</i> , wctype("alpha")) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| iswcntrl(<i>wc</i>) | iswctype(<i>wc</i> , wctype("cntrl")) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| iswdigit(<i>wc</i>) | iswctype(<i>wc</i> , wctype("digit")) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| iswgraph(<i>wc</i>) | iswctype(<i>wc</i> , wctype("graph")) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| iswlower(<i>wc</i>) | iswctype(<i>wc</i> , wctype("lower")) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| iswprint(<i>wc</i>) | iswctype(<i>wc</i> , wctype("print")) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| iswpunct(<i>wc</i>) | iswctype(<i>wc</i> , wctype("punct")) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| iswspace(<i>wc</i>) | iswctype(<i>wc</i> , wctype("space")) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| iswupper(<i>wc</i>) | iswctype(<i>wc</i> , wctype("upper")) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| iswxdigit(<i>wc</i>) | iswctype(<i>wc</i> , wctype("xdigit")) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

`iswctype(wc, wctype("blank"))`

does not have an equivalent `isw*()` function.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |
| CSI | Enabled |

SEE ALSO

iswalphabet(3C), **setlocale(3C)**, **wctype(3C)**, **attributes(5)**, **environ(5)**

| NAME | j0, j1, jn – Bessel functions of the first kind | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lm [library ...] #include <math.h> double j0(double x); double j1(double x); double jn(int n, double x);</pre> | | | | |
| DESCRIPTION | The j0() , j1() and jn() functions compute Bessel functions of x of the first kind of orders 0, 1 and n respectively. | | | | |
| RETURN VALUES | <p>Upon successful completion, j0(), j1() and jn() return the relevant Bessel value of x of the first kind.</p> <p>If the x argument is too large in magnitude, 0 is returned and <code>errno</code> may be set to <code>ERANGE</code>.</p> <p>If x is NaN, NaN is returned.</p> <p>For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.</p> | | | | |
| ERRORS | <p>The j0(), j1() and jn() functions may fail if:</p> <p>ERANGE The value of x was too large in magnitude.</p> | | | | |
| USAGE | An application wishing to check for error situations should set <code>errno</code> to 0 before calling j0() , j1() or jn() . If <code>errno</code> is non-zero on return, or the return value is NaN, an error has occurred. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |

j0(3M)

Mathematical Library

SEE ALSO `isnan(3M)` , `matherr(3M)` , `y0(3M)` , `attributes(5)` , `standards(5)`

NAME kerberos, krb_mk_req, krb_rd_req, krb_kntoln, krb_set_key, krb_get_cred, krb_mk_safe, krb_rd_safe, krb_mk_err, krb_rd_err – Kerberos authentication library

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lkrb
[
library
... ]
#include <kerberos/krb.h>
extern char *krb_err_txt[];

int krb_mk_req(KTEXT authent, const char * service, const char * instance, const char *
realm, const long checksum);

int krb_rd_req(const KTEXT authent, const char * service, char * instance, const long
from_addr, AUTH_DAT * ad, const char * fn);

int krb_kntoln(const AUTH_DAT * ad, char * lname);

int krb_set_key(const char * key, const int cvt);

int krb_get_cred(const char * service, const char * instance, const char * realm,
CREDENTIALS * c);

long krb_mk_safe(const uchar_t * in, uchar_t * out, const ulong_t in_length, const
des_cblock * key, const struct sockaddr_in * sender, const struct sockaddr_in * receiver);

long krb_rd_safe(const uchar_t * in, const ulong_t length, const des_cblock * key, const
struct sockaddr_in * sender, const struct sockaddr_in * receiver, MSG_DAT * msg_data);

long krb_mk_err(uchar_t * out, const long code, const char * string);

long krb_rd_err(const uchar_t * in, const ulong_t length, long * code, MSG_DAT *
msg_data);
```

DESCRIPTION

This library supports network authentication and various related operations. The library contains many routines beyond those described in this man page, but they are not intended to be used directly. Instead, they are called by the routines that are described, the authentication server and the login program.

krb_err_txt[] contains text string descriptions of various Kerberos error codes returned by some of the routines below.

krb_mk_req() takes a pointer to a text structure in which an authenticator is to be built. It also takes the name, instance, and realm of the service to be used and an optional checksum. It is up to the application to decide how to generate the checksum. **krb_mk_req()** then retrieves a ticket for the desired service and creates an authenticator. The authenticator is built in *authent* and is accessible to the calling procedure.

It is up to the application to get the authenticator to the service where it will be read by **krb_rd_req()**. Unless an attacker possesses the session key contained in the ticket, it will be unable to modify the authenticator. Thus, the checksum can be used to verify the authenticity of the other data that will pass through a connection.

krb_mk_req() returns KSUCCESS if successful, otherwise a Kerberos error code as defined in `<kerberos/krb.h>`.

krb_rd_req() takes an authenticator of type KTEXT, a service name, an instance, the address of the host originating the request, and a pointer to a structure of type AUTH_DAT which is filled in with information obtained from the authenticator. It also optionally takes the name of the file in which it will find the secret key(s) for the service. If the supplied *instance* is "", then the first service key with the same service name found in the service key file will be used, and the *instance* argument will be filled in with the chosen instance. This means that the caller must provide space for such an instance name.

If the last argument is the null string (""), **krb_rd_req()** will use the file `/etc/srvtab` to find its keys. If the last argument is NULL, it will assume that the key has been set by **krb_set_key()** and will not bother looking further.

krb_rd_req() is used to find out information about the principal when a request has been made to a service. It is up to the application protocol to get the authenticator from the client to the service. The authenticator is then passed to **krb_rd_req()** to extract the desired information.

krb_rd_req() returns zero (RD_AP_OK) upon successful authentication. If a packet was forged, modified, or replayed, authentication will fail. If the authentication fails, a non-zero value is returned indicating the particular problem encountered. See `<kerberos/krb.h>` for the list of error codes.

krb_kntoln() converts a Kerberos name to a local name. It takes a structure of type AUTH_DAT and uses the name, instance, and realm to determine the corresponding local name. A valid local name is returned if the instance is NULL and the realm is the same as the local realm. The local name returned is the Kerberos name and can be used by an application to change uids, directories, or other parameters. This routine is not an integral part of Kerberos, but is provided to support the use of Kerberos in existing utilities. This routine returns KSUCCESS or KFAILURE.

krb_set_key() takes as an argument a DES key. It then creates a key schedule from it and saves the original key to be used as an initialization vector. It is used to set the server's key which must be used to decrypt tickets.

If called with a non-zero second argument, **krb_set_key()** will first convert the input from a string of arbitrary length to a DES key by encrypting it with a one-way function.

In most cases it should not be necessary to call **krb_set_key()**. The necessary keys will usually be obtained and set inside **krb_rd_req()**. **krb_set_key()** is provided for those applications that do not wish to place the application keys on disk. It returns 0 for success, otherwise a non-zero value.

krb_get_cred() searches the caller's ticket file for a ticket for the given *service*, *instance*, and *realm*. If a ticket is found, the given CREDENTIALS structure is filled in with the ticket information.

If the ticket was found, **krb_get_cred()** returns GC_OK. If the ticket file cannot be found, cannot be read, does not belong to the user (other than root), is not a regular file, or is in the wrong mode, the error GC_TKFIL is returned.

krb_mk_safe() creates an authenticated, but unencrypted message from any arbitrary application data, pointed to by *in* and *in_length* bytes long. The private session key, pointed to by *key*, is used to seed the **quad_cksum()** checksum algorithm used as part of the authentication. *sender* and *receiver* point to the Internet address of the two parties. This message does not provide privacy, but does protect (via detection) against modifications, insertions or replays. The encapsulated message and header are placed in the area pointed to by *out* and the routine returns the length of the output, or -1 indicating an error.

krb_rd_safe() authenticates a received **krb_mk_safe()** message. *in* points to the beginning of the received message, whose length is specified in *in_length*. The private session key, pointed to by *key*, is used to seed the **quad_cksum()** routine as part of the authentication. *msg_data* is a pointer to a MSG_DAT struct, defined in <kerberos/krb.h>. The routine fills in these MSG_DAT fields: the *app_data* field with a pointer to the application data, *app_length* with the length of the *app_data* field, *time_sec* and *time_5ms* with the timestamps in the message, and *swap* with a 1 if the byte order of the receiver is different than that of the sender. (The application must still determine if it is appropriate to byte-swap application data; the Kerberos protocol fields are already taken care of.)

The routine returns zero if successful, or a Kerberos error code. Modified messages and old messages cause errors, but it is up to the caller to check the time sequence of messages, and to check against recently replayed messages.

krb_mk_err() constructs an application level error message that may be used along with **krb_mk_safe()**. *out* is a pointer to the output buffer, *code* is an application specific error code, and *string* is an application specific error string. This routine returns the length of the error reply.

krb_rd_err() unpacks a received **krb_mk_err()** message. *in* points to the beginning of the received message, whose length is specified in *in_length*. *code* is a pointer to a value to be filled in with the error value provided by the application. *msg_data* is a pointer to a MSG_DAT struct, defined in <kerberos/krb.h>. The routine fills in these MSG_DAT fields: the *app_data* field with a pointer to the application error text, *app_length* with the length of the *app_data* field, and *swap* with a 1 if the byte order of the receiver is different than that of the sender. (The application must still determine if it is appropriate to byte-swap application data; the Kerberos protocol fields are already taken care of).

The routine returns zero if the error message has been successfully received, or a Kerberos error code.

The KTEXT structure is used to pass around text of varying lengths. It consists of a buffer for the data, and a length. **krb_rd_req()** takes an argument of this type containing the authenticator, and **krb_mk_req()** returns the authenticator in a structure of this type. KTEXT itself is really a pointer to the structure. The actual structure is of type KTEXT_ST.

The AUTH_DAT structure is filled in by **krb_rd_req()**. It must be allocated before calling **krb_rd_req()**, and a pointer to it is passed. The structure is filled in with data obtained from Kerberos. The MSG_DAT structure is filled in by either **krb_rd_safe()** or **krb_rd_err()**. It must be allocated before the call and a pointer to it is passed. The structure is filled in with data obtained from Kerberos.

FILES

/usr/lib/libkrb.*
 /etc/aname
 /etc/srvtab
 /tmp/tkt *uid*

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO | `kerberos(1)` , `kerberos_rpc(3N)` , `krb_realmofhost(3N)` ,
`krb_sendauth(3N)` , `krb_set_tkt_string(3N)` , `krb.conf(4)` ,
`krb.realms(4)` , `attributes(5)`

NOTES | These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

BUGS | The caller of `krb_rd_req()` and `krb_rd_safe()` must check time order and for replay attempts.

AUTHORS | Clifford Neuman, MIT Project Athena Steve Miller, MIT Project Athena/Digital Equipment Corporation

RESTRICTIONS | COPYRIGHT 1985,1986,1989 Massachusetts Institute of Technology

| | |
|--------------------|--|
| NAME | kerberos_rpc, authkerb_getucred, authkerb_seccreate, svc_kerb_reg – library routines for remote procedure calls using Kerberos authentication |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lkrb [<i>library</i> ...] #include <rpc/rpc.h> #include <sys/types.h> int authkerb_getucred(const struct svc_req * <i>rqst</i>, uid_t * <i>uidp</i>, gid_t * <i>gidp</i>, short * <i>gidlenp</i>, int <i>gidlist</i> [NGROUPS]); AUTH * authkerb_seccreate(const char * <i>service</i>, const char * <i>srv_inst</i>, const char * <i>realm</i>, const uint_t <i>window</i>, const char * <i>timehost</i>, int * <i>status</i>); int svc_kerb_reg(const SVCXPRT * <i>xprt</i>, const char * <i>name</i>, const char * <i>inst</i>, const char * <i>realm</i>);</pre> |
| DESCRIPTION | <p>RPC library routines allow C programs to make procedure calls on other machines across the network.</p> <p>RPC supports various authentication flavors. Among them are:</p> <p>AUTH_NONE (none) no authentication.</p> <p>AUTH_SYS Traditional UNIX-style authentication.</p> <p>AUTH_DES DES encryption-based authentication.</p> <p>AUTH_KERB Kerberos encryption-based authentication.</p> <p>The authkerb_getucred() , authkerb_seccreate() , and svc_kerb_reg() routines implement the AUTH_KERB authentication flavor. The kerbd daemon (see kerbd(1M)) must be running for the AUTH_KERB authentication system to work for kernel based services such as NFS, and kinit(1) must have been run by the user in all cases. Only the AUTH_KERB style of authentication is discussed here. For information about the AUTH_NONE and AUTH_SYS styles of authentication, refer to rpc_clnt_auth(3N) . For information about the AUTH_DES style of authentication, refer to secure_rpc(3N) .</p> |

Routines

See `rpc(3N)` for the definition of the `AUTH` data structure.

```
int authkerb_getucred(const struct svc_req * rqst , uid_t * uidp , gid_t * gidp ,
short * gidlenp , int gidlist [NGROUPS]);
```

authkerb_getucred() is used on the server side for converting an `AUTH_KERB` credential received in an RPC request, which is operating system independent, into an `AUTH_SYS` credential. This routine returns 1 if it succeeds, 0 if it fails.

* `uidp` is set to the numerical ID of the user associated with the RPC request referenced by `rqst`. * `gidp` is set to the numerical ID of the user's group. The numerical IDs of the other groups to which the user belongs are stored in `gidlist []`. * `gidlenp` is set to the number of valid group ID entries returned in `gidlist []`. All information returned by this routine is based on the Kerberos principal name contained in `rqst`. This principal name is taken to be the login name of the user, and the IDs returned are the same as if that user had physically logged in to the system.

```
AUTH *authkerb_seccreate(const char * service , const char * srv_inst , const
char * realm , const uint_t window , const char * timehost , int * status );
```

authkerb_seccreate() is used on the client side to return an authentication handle that will enable the use of the Kerberos authentication system. The first parameter `service` is the Kerberos principal name of the service to be used. This name is generally a constant with respect to the service being used. `srv_instance` is the instance of the service to be called, and may be `NULL` to indicate any instance. `realm` is the Kerberos realm name of the desired service. If it is `NULL`, then the local default realm will be used.

The fourth parameter is the `window` on the validity of the client credential, given in seconds. If the difference in time between the client's clock and the server's clock exceeds `window`, the server will reject the client's credentials, and the clock will have to be resynchronized. A small window is more secure than a large one, but choosing too small of a window will increase the frequency of resynchronizations because of clock drift.

The fifth parameter, `timehost`, is optional. If it is `NULL`, then the authentication system will assume that the local clock is always in sync with the `timehost` clock, and will not attempt resynchronizations. If a `timehost` is supplied, however, then the system will consult with the remote time service whenever resynchronization is required. This parameter is usually the name of the host on which the server is running.

The final parameter `status` is also optional. If `status` is supplied, then it will be used to return a Kerberos error status codes if an error occurs. If `status` is `NULL`, then no detailed error codes will be returned.

If `authkerb_seccreate()` fails, it returns `NULL`.

```
int svc_kerb_reg(const SVCXPRT * xprt , const char * name , const char * inst
, const char * realm );
```

`svc_kerb_reg()` performs registration tasks in the server which are required before `AUTH_KERB` requests can be processed. *xprt* is the RPC transport to which this information is to be associated. If *xprt* is `NULL` then this registration will be effective for any requests arriving on transports that have not been specifically registered. The service handles associated with connection endpoints are not exposed to the programmer. Consequently, *xprt* should be `NULL` for connection-oriented transports.

The other parameters describe the Kerberos principal identity that this server will take on. This must be the same identity that the clients will use when requesting Kerberos tickets for authentication. *name* is the principal name of the service and must be provided. *inst* is the instance. This parameter may be `NULL` to specify the `NULL` instance of the service. Most common would be for *inst* to be "*" which allows the Kerberos library to determine the correct instance to use, such as the hostname that the service is running on. *realm* is the Kerberos realm name to use in validating tickets. If it is `NULL`, then the local default realm will be used.

`svc_kerb_reg()` should generally be called immediately before `svc_run()`. It returns 0 if it succeeds, and -1 if it fails.

ATTRIBUTES

See `attributes` (5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`kerberos(1)`, `kinit(1)`, `kerbd(1M)`, `rpc(3N)`, `rpc_clnt_auth(3N)`, `secure_rpc(3N)`, `svc_run(3N)` `attributes(5)`

NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

| NAME | keyname, key_name – return character string used as key name | | | | | | | | | | | | |
|---|--|-------|--------------------|-------------------|--------------------|-------------------|------------|---|-------------|---|------------------|-------------------|-------------|
| SYNOPSIS | <pre>#include <curses.h> char * keyname(int c); char * key_name(wchar_t wc);</pre> | | | | | | | | | | | | |
| PARAMETERS | <p>c Is an 8 bit-character or a key code.</p> <p>wc Is a wide character key name.</p> | | | | | | | | | | | | |
| DESCRIPTION | <p>The keyname() function returns a string pointer to the key name. Make a duplicate copy of the returned string if you plan to modify it.</p> <p>The key_name() function is similar except that it accepts a wide character key name.</p> <p>The following table shows the format of the key name based on the input.</p> <table border="1"> <thead> <tr> <th>Input</th> <th>Format of Key Name</th> </tr> </thead> <tbody> <tr> <td>Visible character</td> <td>The same character</td> </tr> <tr> <td>Control character</td> <td>^ <i>X</i></td> </tr> <tr> <td>Meta-character (keyname() only)</td> <td>M- <i>X</i></td> </tr> <tr> <td>Key value defined in <code><curses.h></code> (keyname() only)</td> <td><i>KEY_ name</i></td> </tr> <tr> <td>None of the above</td> <td>UNKNOWN KEY</td> </tr> </tbody> </table> <p>In the preceding table, <i>X</i> can be either a visible character with the high bit cleared or a control character.</p> | Input | Format of Key Name | Visible character | The same character | Control character | ^ <i>X</i> | Meta-character (keyname() only) | M- <i>X</i> | Key value defined in <code><curses.h></code> (keyname() only) | <i>KEY_ name</i> | None of the above | UNKNOWN KEY |
| Input | Format of Key Name | | | | | | | | | | | | |
| Visible character | The same character | | | | | | | | | | | | |
| Control character | ^ <i>X</i> | | | | | | | | | | | | |
| Meta-character (keyname() only) | M- <i>X</i> | | | | | | | | | | | | |
| Key value defined in <code><curses.h></code> (keyname() only) | <i>KEY_ name</i> | | | | | | | | | | | | |
| None of the above | UNKNOWN KEY | | | | | | | | | | | | |
| RETURN VALUES | On success, these functions return a pointer to the string used as the key's name. Otherwise, they return a null pointer. | | | | | | | | | | | | |
| ERRORS | None. | | | | | | | | | | | | |
| SEE ALSO | meta (3XC) | | | | | | | | | | | | |

| | | | | | |
|----------------------|--|-------------------|--|------------------|--------------------------|
| NAME | keypad – enable/disable keypad handling | | | | |
| SYNOPSIS | <pre>#include <curses.h> int keypad(WINDOW *win, bool bf);</pre> | | | | |
| PARAMETERS | <table><tr><td><i>win</i></td><td>Is a pointer to the window in which to enable keypad handling.</td></tr><tr><td><i>bf</i></td><td>Is a Boolean expression.</td></tr></table> | <i>win</i> | Is a pointer to the window in which to enable keypad handling. | <i>bf</i> | Is a Boolean expression. |
| <i>win</i> | Is a pointer to the window in which to enable keypad handling. | | | | |
| <i>bf</i> | Is a Boolean expression. | | | | |
| DESCRIPTION | <p>If <i>bf</i> is TRUE, getch(3XC) handles special keys from the keyboard on the terminal associated with <i>win</i> as single values instead of character sequences. For example, if the user presses the right arrow key, getch() returns a single value, <code>KEY_RIGHT</code>, that represents the function key; otherwise, X/Open Curses handles the special keys as normal text.</p> <p>See getch() for a list of tokens for function keys that are returned by getch().</p> | | | | |
| RETURN VALUES | On success, the keypad() function returns <code>OK</code> . Otherwise, it returns <code>ERR</code> . | | | | |
| ERRORS | None. | | | | |
| SEE ALSO | getch(3XC) | | | | |

| | |
|----------------------|---|
| NAME | killpg – send signal to a process group |
| SYNOPSIS | #include <signal.h> int killpg (pid_t <i>pgrp</i> , int <i>sig</i>); |
| DESCRIPTION | The killpg() function sends the signal <i>sig</i> to the process group <i>pgrp</i> . See signal(5) for a list of signals. The real or effective user ID of the sending process must match the real or saved set-user ID of the receiving process, unless the effective user ID of the sending process is the privileged user. A single exception is the signal SIGCONT, which may always be sent to any descendant of the current process. |
| RETURN VALUES | Upon successful completion, 0 is returned. Otherwise, -1 is returned and errno is set to indicate the error. |
| ERRORS | The killpg() function will fail and no signal will be sent if: EINVAL The <i>sig</i> argument is not a valid signal number. EPERM The effective user ID of the sending process is not privileged user, and neither its real nor effective user ID matches the real or saved set-user ID of one or more of the target processes. ESRCH No processes were found in the specified process group. |
| SEE ALSO | kill(2) , setpgrp(2) , sigaction(2) , signal(5) |

| | |
|--------------------|--|
| NAME | krb_realmofhost, krb_get_phost, krb_get_krbhst, krb_get_admhst, krb_get_lrealm – additional Kerberos utility routines |
| SYNOPSIS | <pre> cc [flag ...] file ... -lkrb [library ...] #include <kerberos/krb.h> #include <netinet/in.h> char *krb_realmofhost(const char * host); char *krb_get_phost(const char * alias); int krb_get_krbhst(char * host, const char * realm, const int n); int krb_get_admhst(char * host, const char * realm, const int n); int krb_get_lrealm(char * realm, const int n); </pre> |
| DESCRIPTION | <p>krb_realmofhost() returns the Kerberos realm of the host <i>host</i>, as determined by the translation table <code>/etc/krb.realms</code>. <i>host</i> should be the fully-qualified domain-style primary host name of the host in question. In order to prevent certain security attacks, this routine must either have a prior knowledge of a host's realm, or obtain such information securely.</p> <p>The format of the translation file is described by <code>krb.realms(4)</code>. If <i>host</i> exactly matches a <code>host_name</code> line, the corresponding realm is returned. Otherwise, if the domain portion of <i>host</i> matches a <code>domain_name</code> line, the corresponding realm is returned. If <i>host</i> contains a domain, but no translation is found, <i>host</i>'s domain is converted to upper-case and returned. If <i>host</i> contains no discernible domain, or an error occurs, the local realm name, as supplied by <code>krb_get_lrealm()</code>, is returned.</p> <p>krb_get_phost() converts the hostname <i>alias</i> (which can be either an official name or an alias) into the instance name to be used in obtaining Kerberos tickets for most services, including the Berkeley rcmd suite (<code>rlogin</code>, <code>rcp</code>, <code>rsh</code>). The current convention is to return the first segment of the official domain-style name after conversion to lower case.</p> <p>krb_get_krbhst() fills in <i>host</i> with the hostname of the <i>n</i>th host running a Kerberos key distribution center (KDC) for realm <i>realm</i>, as specified in the</p> |

configuration file (`/etc/krb.conf` or `krb.conf` NIS map). The configuration format is described by `krb.conf(4)`. If the host is successfully filled in, the routine returns `KSUCCESS`. If the file (or NIS map) cannot be accessed, and *n* equals 1, then the hostname `kerberos` is filled in, and `KSUCCESS` is returned. If there are fewer than *n* hosts running a Kerberos KDC for the requested realm, or the configuration file is malformed, the routine returns `KFAILURE`.

When there is both a local `/etc/krb.conf` and a `krb.conf` NIS map, then the entries are counted starting first with the local file, then continuing with the NIS map. For example, if the local `/etc/krb.conf` file contains two entries which match *realm*, and the NIS map contains one matching entry, then there are three possible matches that `krb_get_krbhst()` can return. The first two (for *n* values 1 and 2) come from the file, and the third (for *n* equal to 3) comes from the map.

`krb_get_admhst()` fills in *host* with the hostname of the *n*th host running a Kerberos KDC database administration server for realm *realm*, as specified in `/etc/krb.conf`. If the file cannot be opened or is malformed, or there are fewer than *n* hosts running a Kerberos KDC database administration server, the routine returns `KFAILURE`.

The character arrays used as return values for `krb_get_krbhst()` and `krb_get_admhst()` should be large enough to hold any hostname.

`krb_get_lrealm()` fills in *realm* with the *n*th realm of the local host, as specified in the configuration file. *realm* should be at least `REALM_SZ` (from `<kerberos/krb.h>`) characters long. The return value is either `KSUCCESS` or `KFAILURE`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`kerberos(3N)`, `krb.conf(4)`, `krb.realms(4)`, `attributes(5)`

FILES

| | |
|------------------------------|---|
| <code>/etc/krb.realms</code> | translation file for host-to-realm mapping. |
| <code>/etc/krb.conf</code> | local realm-name and realm/server configuration file. |

NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

BUGS

The current convention for instance names is too limited; the full domain name should be used.

krb_get_lrealm() currently only supports n equal to 1. It should really consult the user's ticket cache to determine the user's current realm, rather than consulting a file on the host.

| | |
|-----------------------|---|
| NAME | krb_sendauth, krb_recvauth, krb_net_write, krb_net_read – Kerberos routines for sending authentication via network stream sockets |
| SYNOPSIS | <pre>cc [flag ...] file ... -lkrb [library ...] #include <kerberos/krb.h> #include <netinet/in.h> int krb_sendauth(const long options, const int fd, KTEXT ktext, const char * service, const char * inst, const char * realm, const ulong_t checksum, MSG_DAT * msg_data, CREDENTIALS * cred, Key_schedule schedule, const struct sockaddr_in * laddr, const struct sockaddr_in * faddr, const char * version); int krb_recvauth(const long options, const int fd, KTEXT ktext, const char * service, char * inst, const struct sockaddr_in * faddr, const struct sockaddr_in * laddr, AUTH_DAT * auth_data, const char * filename, Key_schedule schedule, char * version); int krb_net_write(const int fd, const char * buf, const int len); int krb_net_read(const int fd, char * buf, const int len);</pre> |
| DESCRIPTION | <p>These functions, which are built on top of the core Kerberos library, provide a convenient means for client and server programs to send authentication messages to one another through network connections.</p> <p>The krb_sendauth() function sends an authenticated ticket from the client program to the server program by writing the ticket to a network socket.</p> <p>The krb_recvauth() function receives the ticket from the client by reading from a network socket.</p> |
| krb_sendauth() | <p>This function writes the ticket to the network socket specified by the file descriptor <i>fd</i>, returning KSUCCESS if the write proceeds successfully, and an error code if it does not.</p> <p>The <i>ktext</i> argument should point to an allocated KTEXT_ST structure. The <i>service</i>, <i>inst</i>, and <i>realm</i> arguments specify the server program's Kerberos principal name, instance, and realm. If you are writing a client that uses the local realm exclusively, you can set the <i>realm</i> argument to NULL.</p> |

**krb_sendauth() and
Mutual
Authentication****krb_recvauth()**

The *version* argument allows the client program to pass an application-specific version string that the server program can then match against its own version string. The *version* string can be up to KSEND_VNO_LEN (see <kerberos/krb.h>) characters in length.

The *checksum* argument can be used to pass checksum information to the server program. The client program is responsible for specifying this information. This checksum information is difficult to corrupt because **krb_sendauth()** passes it over the network in encrypted form. The *checksum* argument is passed as the *checksum* argument to **krb_mk_req()** (see **kerberos(3N)**).

You can set **krb_sendauth()**'s other arguments to NULL unless you want the client and server programs to mutually authenticate themselves. In the case of mutual authentication, the client authenticates itself to the server program, and demands that the server in turn authenticate itself to the client.

If you want mutual authentication, make sure that you read all pending data from the local socket before calling **krb_sendauth()**. Set **krb_sendauth()**'s *options* argument to KOPT_DO_MUTUAL (this macro is defined in <kerberos/krb.h>); make sure that the *laddr* argument points to the address of the local socket, and that *faddr* points to the foreign socket's network address.

krb_sendauth() fills in the other arguments — *msg_data*, *cred*, and *schedule* — before sending the ticket to the server program. You must, however, allocate space for these arguments before calling the function.

krb_sendauth() supports two other options: KOPT_DONT_MK_REQ and KOPT_DONT_CANON. If called with *options* set as KOPT_DONT_MK_REQ, **krb_sendauth()** will not use the **krb_mk_req()** (see **kerberos(3N)**) function to retrieve the ticket from the Kerberos server. The *ktext* argument must point to an existing ticket and authenticator (such as would be created by **krb_mk_req()**), and the *service*, *inst*, and *realm* arguments can be set to NULL.

If called with *options* set as KOPT_DONT_CANON, **krb_sendauth()** will not convert the service's instance to canonical form using **krb_get_phost()** (see **krb_realmofhost(3N)**).

If you want to call **krb_sendauth()** with a multiple *options* specification, construct *options* as a bitwise-OR of the options you want to specify.

The **krb_recvauth()** function reads a ticket/authenticator pair from the socket pointed to by the *fd* argument. Set the *options* argument as a bitwise-OR of the options desired. Currently only KOPT_DO_MUTUAL is useful to the receiver.

The *ktext* argument should point to an allocated KTEXT_ST structure. **krb_recvauth()** fills *ktext* with the ticket/authenticator pair read from *fd*, then passes it to **krb_rd_req()** (see **kerberos(3N)**).

The *service* and *inst* arguments specify the expected service and instance for which the ticket was generated. They are also passed to **krb_rd_req()** (see **kerberos(3N)**). The *inst* argument may be set to "*" if the caller wishes **krb_mk_req()** (see **kerberos(3N)**) to fill in the instance used (note that there must be space in the *inst* argument to hold a full instance name, see **krb_mk_req()** on **kerberos(3N)**).

The *faddr* argument should point to the address of the peer which is presenting the ticket. It is also passed to **krb_rd_req()** (see **kerberos(3N)**).

If the client and server plan to mutually authenticate one another, the *laddr* argument should point to the local address of the file descriptor. Otherwise you can set this argument to NULL.

The *auth_data* argument should point to an allocated AUTH_DAT area. It is passed to and filled in by **krb_rd_req()** (see **kerberos(3N)**). The checksum passed to the corresponding **krb_sendauth()** is available as part of the filled-in AUTH_DAT area.

The *filename* argument specifies the filename which the service program should use to obtain its service key. **krb_recvauth()** passes *filename* to the **krb_rd_req()** function, see **kerberos(3N)**. If you set this argument to "", **krb_rd_req()** looks for the service key in the file */etc/srvtab*.

If the client and server are performing mutual authentication, the *schedule* argument should point to an allocated Key_schedule. Otherwise it is ignored and may be NULL.

The *version* argument should point to a character array of at least KSEND_VNO_LEN characters. It is filled in with the version string passed by the client to **krb_sendauth()**.

krb_net_write() and krb_net_read()

The **krb_net_write()** function emulates the **write(2)** system call, but guarantees that all data specified is written to *fd* before returning, unless an error condition occurs.

The **krb_net_read()** function emulates the **read(2)** system call, but guarantees that the requested amount of data is read from *fd* before returning, unless an error condition occurs.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

| | |
|---------------------|--|
| SEE ALSO | <code>read(2)</code> , <code>write(2)</code> , <code>kerberos(3N)</code> , <code>kerberos_rpc(3N)</code> , <code>krb_realmofhost(3N)</code> , <code>attributes (5)</code> |
| NOTES | These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread. |
| BUGS | <code>krb_sendauth()</code> , <code>krb_recvauth()</code> , <code>krb_net_write()</code> , and <code>krb_net_read()</code> will not work properly on sockets set to non-blocking I/O mode. |
| AUTHOR | John T. Kohl, MIT Project Athena |
| RESTRICTIONS | Copyright 1988, Massachusetts Institute of Technology. For copying and distribution information, please see the header <code><kerberos/mit-copyright.h></code> . |

NAME krb_set_tkt_string – set Kerberos ticket cache file name

SYNOPSIS

```
cc [ flag ... ] file ... -lkrb [ library ... ]
#include <kerberos/krb.h>
```

```
void krb_set_tkt_string(const char *filename);
```

DESCRIPTION **krb_set_tkt_string()** sets the name of the file that holds the user's cache of Kerberos server tickets and associated session keys.

The string *filename* passed in is copied into local storage. Only MAXPATHLEN-1 (see <sys/param.h>) characters of the filename are copied in for use as the cache file name.

This routine should be called during initialization, before other Kerberos routines are called; otherwise the routines which fetch the ticket cache file name may be called and return an undesired ticket file name until this routine is called.

FILES

| | |
|---------------------|--|
| /tmp/tkt <i>uid</i> | default ticket file name, unless the environment variable KRBTKFILE is set. <i>uid</i> denotes the user's uid, in decimal. |
|---------------------|--|

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO **kerberos(3N)**, **attributes(5)**

NOTES This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

| | |
|------------------------|---|
| NAME | kstat – kernel statistics facility |
| DESCRIPTION | The <code>kstat</code> facility is a general-purpose mechanism for providing kernel statistics to users. |
| The kstat model | The kernel maintains a linked list of statistics structures, or <code>kstats</code> . Each <code>kstat</code> has a common header section and a type-specific data section. The header section is defined by the <code>kstat_t</code> structure: |
| kstat header | <pre> typedef int kid_t; /* unique kstat id */ typedef struct kstat { /* * Fields relevant to both kernel and user */ hrtime_t ks_crttime; /* creation time */ struct kstat *ks_next; /* kstat chain linkage */ kid_t ks_kid; /* unique kstat ID */ char ks_module[KSTAT_STRLEN]; /* module name */ uchar_t ks_resv; /* reserved */ int ks_instance; /* module's instance */ char ks_name[KSTAT_STRLEN]; /* kstat name */ uchar_t ks_type; /* kstat data type */ char ks_class[KSTAT_STRLEN]; /* kstat class */ uchar_t ks_flags; /* kstat flags */ void *ks_data; /* kstat type-specific data */ uint_t ks_ndata; /* # of data records */ size_t ks_data_size; /* size of kstat data section */ hrtime_t ks_snaptime; /* time of last data snapshot */ /* * Fields relevant to kernel only */ int (*ks_update)(struct kstat *, int); void *ks_private; int (*ks_snapshot)(struct kstat *, void *, int); void *ks_lock; } kstat_t; </pre> <p>The fields that are of significance to the user are:</p> <p><code>ks_crttime</code> The time the <code>kstat</code> was created. This allows you to compute the rates of various counters since the <code>kstat</code> was created; "rate since boot" is replaced by the more general concept of "rate since <code>kstat</code> creation". All times associated with <code>kstats</code> (such as creation time, last snapshot time, <code>kstat_timer_t</code> and <code>kstat_io_t</code> timestamps, and the like) are 64-bit nanosecond values. The accuracy of <code>kstat</code> timestamps is machine dependent, but the precision (units) is the same</p> |

| | |
|--|---|
| | across all platforms. See <code>gethrtime(3C)</code> for general information about high-resolution timestamps. |
| <code>ks_next</code> | kstats are stored as a linked list, or chain. <code>ks_next</code> points to the next kstat in the chain. |
| <code>ks_kid</code> | A unique identifier for the kstat. |
| <code>ks_module</code> , <code>ks_instance</code> | contain the name and instance of the the module that created the kstat. In cases where there can only be one instance, <code>ks_instance</code> is 0. |
| <code>ks_name</code> | gives a meaningful name to a kstat. The full kstat namespace is <code><ks_module,ks_instance,ks_name></code> , so the name only need be unique within a module. |
| <code>ks_type</code> | The type of data in this kstat. kstat data types are discussed below. |
| <code>ks_class</code> | Each kstat can be characterized as belonging to some broad class of statistics, such as disk, tape, net, vm, and streams. This field can be used as a filter to extract related kstats. The following values are currently in use: <code>disk</code> , <code>tape</code> , <code>controller</code> , <code>net</code> , <code>rpc</code> , <code>vm</code> , <code>kvm</code> , <code>hat</code> , <code>streams</code> , <code>kmem</code> , <code>kmem_cache</code> , <code>kstat</code> , and <code>misc</code> . (The kstat class encompasses things like <i>kstat_types</i> .) |
| <code>ks_data</code> , <code>ks_ndata</code> , <code>ks_data_size</code> | <code>ks_data</code> is a pointer to the kstat's data section. The type of data stored there depends on <code>ks_type</code> . <code>ks_ndata</code> indicates the number of data records. Only some kstat types support multiple data records. Currently, <code>KSTAT_TYPE_RAW</code> , <code>KSTAT_TYPE_NAMED</code> and <code>KSTAT_TYPE_TIMER</code> kstats support multiple data records. <code>KSTAT_TYPE_INTR</code> and <code>KSTAT_TYPE_IO</code> kstats support only one data record. <code>ks_data_size</code> is the total size of the data section, in bytes. |
| <code>ks_snaptime</code> | The timestamp for the last data snapshot. This allows you to compute activity rates: |

```
rate = (new_count - old_count) / (new_snaptime - old_snaptime);
```

kstat data types

The following types of kstats are currently available:

```
#define KSTAT_TYPE_RAW      0    /* can be anything */
#define KSTAT_TYPE_NAMED  1    /* name/value pairs */
#define KSTAT_TYPE_INTR   2    /* interrupt statistics */
#define KSTAT_TYPE_IO     3    /* I/O statistics */
#define KSTAT_TYPE_TIMER  4    /* event timers */
```

To get a list of all kstat types currently supported in the system, tools can read out the standard system kstat *kstat_types* (full name spec is <"unix", 0, "kstat_types">). This is a KSTAT_TYPE_NAMED kstat in which the *name* field describes the type of kstat, and the *value* field is the kstat type number (for example, KSTAT_TYPE_IO is type 3 – see above).

Raw kstat

KSTAT_TYPE_RAW raw data

The "raw" kstat type is just treated as an array of bytes. This is generally used to export well-known structures, like *sysinfo*.

Name=value kstat

KSTAT_TYPE_NAMED A list of arbitrary *name=value* statistics.

```
typedef struct kstat_named {
    char name[KSTAT_STRLEN];          /* name of counter */
    uchar_t data_type;                /* data type */
    union {
        char c[16];                  /* enough for 128-bit ints */
        int32_t i32;
        uint32_t ui32;
        int64_t i64;
        uint64_t ui64;

        /* These structure members are obsolete */

        int32_t l;
        uint32_t ul;
        int64_t ll;
        uint64_t ull;
    } value;                          /* value of counter */
} kstat_named_t;
#define KSTAT_DATA_CHAR      0
#define KSTAT_DATA_INT32    1
#define KSTAT_DATA_UINT32   2
#define KSTAT_DATA_INT64    3
```

```
#define KSTAT_DATA_UINT64      4

/* These types are obsolete */

#define KSTAT_DATA_LONG       1
#define KSTAT_DATA_ULONG      2
#define KSTAT_DATA_LONGLONG   3
#define KSTAT_DATA_ULONGLONG  4
#define KSTAT_DATA_FLOAT      5
#define KSTAT_DATA_DOUBLE     6
```

Interrupt kstat

KSTAT_TYPE_INTR **Interrupt statistics.**

An interrupt is a hard interrupt (sourced from the hardware device itself), a soft interrupt (induced by the system via the use of some system interrupt source), a watchdog interrupt (induced by a periodic timer call), spurious (an interrupt entry point was entered but there was no interrupt to service), or multiple service (an interrupt was detected and serviced just prior to returning from any of the other types).

```
#define KSTAT_INTR_HARD      0
#define KSTAT_INTR_SOFT     1
#define KSTAT_INTR_WATCHDOG  2
#define KSTAT_INTR_SPURIOUS  3
#define KSTAT_INTR_MULTSVC   4
#define KSTAT_NUM_INTRS     5

typedef struct kstat_intr {
    uint_t intrs[KSTAT_NUM_INTRS]; /* interrupt counters */
} kstat_intr_t;
```

Event timer kstat

KSTAT_TYPE_TIMER **Event timer statistics.**

These provide basic counting and timing information for any type of event.

```
typedef struct kstat_timer {
    char name[KSTAT_STRLEN]; /* event name */
    uchar_t resv; /* reserved */
    u_longlong_t num_events; /* number of events */
    hrttime_t elapsed_time; /* cumulative elapsed time */
    hrttime_t min_time; /* shortest event duration */
    hrttime_t max_time; /* longest event duration */
    hrttime_t start_time; /* previous event start time */
    hrttime_t stop_time; /* previous event stop time */
} kstat_timer_t;
```

I/O kstat

KSTAT_TYPE_IO

I/O statistics.

```

typedef struct kstat_io {
/*
 * Basic counters.
 */
u_longlong_t nread; /* number of bytes read */
u_longlong_t nwritten; /* number of bytes written */
uint_t reads; /* number of read operations */
uint_t writes; /* number of write operations */
/*
 * Accumulated time and queue length statistics.
 *
 * Time statistics are kept as a running sum of "active" time.
 * Queue length statistics are kept as a running sum of the
 * product of queue length and elapsed time at that length -
 * that is, a Riemann sum for queue length integrated against time.
 *
 *      ^
 *      |
 *      8      |-----|
 *      |      | i4   |
 *      |      | |   |
 * Queue 6    | |   |
 Length      |-----|
 *      4      | i2   |
 *      |      | |   |
 *      |      | i3   |
 *      2-----|
 *      |      | i1   |
 *      |-----|
 *      Time-> t1 t2 t3 t4
 *
 * At each change of state (entry or exit from the queue),
 * we add the elapsed time (since the previous state change)
 * to the active time if the queue length was non-zero during
 * that interval; and we add the product of the elapsed time
 * times the queue length to the running length*time sum.
 *
 * This method is generalizable to measuring residency
 * in any defined system: instead of queue lengths, think
 * of "outstanding RPC calls to server X".
 *
 * A large number of I/O subsystems have at least two basic
 * "lists" of transactions they manage: one for transactions
 * that have been accepted for processing but for which processing
 * has yet to begin, and one for transactions which are actively
 * being processed (but not done). For this reason, two cumulative
 * time statistics are defined here: pre-service (wait) time,
 * and service (run) time.
 *
 * The units of cumulative busy time are accumulated nanoseconds.
 * The units of cumulative length*time products are elapsed time
 * times queue length.
 */
hrtime_t wtime;          /* cumulative wait (pre-service) time */
hrtime_t wlentime;      /* cumulative wait length*time product */

```

(continued)

(Continuation)

```

hrtime_t wlastupdate; /* last time wait queue changed */
hrtime_t rtime;      /* cumulative run (service) time */
hrtime_t rlltime;   /* cumulative run length*time product */
hrtime_t rlastupdate; /* last time run queue changed */
uint_t wcnt;       /* count of elements in wait state */
uint_t rcnt;      /* count of elements in run state */
} kstat_io_t;

```

Using libkstat

The kstat library, `libkstat`, defines the user interface (API) to the system's kstat facility.

You begin by opening `libkstat` with `kstat_open(3K)`, which returns a pointer to a fully initialized kstat control structure. This is your ticket to subsequent `libkstat` operations:

```

typedef struct kstat_ctl {
    kid_t      kc_chain_id; /* current kstat chain ID */
    kstat_t    *kc_chain;  /* pointer to kstat chain */
    int        kc_kd;      /* /dev/kstat descriptor */
} kstat_ctl_t;

```

Only the first two fields, `kc_chain_id` and `kc_chain`, are of interest to `libkstat` clients. (`kc_kd` is the descriptor for `/dev/kstat`, the kernel statistics driver. `libkstat` functions are built on top of `/dev/kstat` `ioctl(2)` primitives. Direct interaction with `/dev/kstat` is strongly discouraged, since it is *not* a public interface.)

`kc_chain` points to your copy of the kstat chain. You typically walk the chain to find and process a certain kind of kstat. For example, to display all I/O kstats:

```

kstat_ctl_t    *kc;
kstat_t        *ksp;
kstat_io_t     kio;

kc = kstat_open();
for (ksp = kc->kc_chain; ksp != NULL; ksp = ksp->ks_next) {
    if (ksp->ks_type == KSTAT_TYPE_IO) {
        kstat_read(kc, ksp, &kio);
        my_io_display(kio);
    }
}

```


| | |
|----------------------|---|
| NAME | kstat_chain_update – update the kstat header chain |
| SYNOPSIS | <pre>cc [flag ...] file ... -lkstat [library ...] #include <kstat.h></pre> <pre>kid_t *kstat_chain_update(kstat_ctl_t *kc);</pre> |
| DESCRIPTION | <p>kstat_chain_update() brings the user's kstat header chain in sync with the kernel's. The kstat chain is a linked list of kstat headers (<i>kstat_t</i>'s), pointed to by <i>kc->kc_chain</i>, which is initialized by kstat_open(3K). This chain constitutes a list of all kstats currently in the system. During normal operation, the kernel will occasionally create new kstats and delete old ones, as various device instances come and go. When this happens, the user's copy of the kstat chain becomes out of date.</p> <p>kstat_chain_update() detects this by comparing the kernel's current kstat chain ID (KCID), which is incremented every time the kstat chain changes, to the user's KCID, <i>kc->kc_chain_id</i>. If the KCID's match, kstat_chain_update() does nothing. Otherwise, it deletes any invalid kstat headers from the user's kstat chain, and adds any new ones, and sets <i>kc->kc_chain_id</i> to the new KCID. All other kstat headers in the user's kstat chain are unmodified.</p> |
| RETURN VALUES | kstat_chain_update() returns the new KCID if the kstat chain has changed, 0 if it hasn't, or -1 on failure. |
| FILES | /dev/kstat kernel statistics driver |
| SEE ALSO | kstat(3K) , kstat_close(3K) , kstat_data_lookup(3K) , kstat_lookup(3K) , kstat_open(3K) , kstat_read(3K) , kstat_write(3K) |

| | |
|----------------------|--|
| NAME | kstat_lookup, kstat_data_lookup – find a kstat by name |
| SYNOPSIS | <pre>cc [flag ...] file ... -lkstat [library ...] #include <kstat.h> kstat_t *kstat_lookup(kstat_ctl_t *kc, char *ks_module, int ks_instance, char *ks_name); void *kstat_data_lookup(kstat_t *ksp, char *name);</pre> |
| DESCRIPTION | <p>kstat_lookup() walks down the kstat chain, <i>kc->kc_chain</i>, looking for a kstat with the same <i>ks_module</i>, <i>ks_instance</i>, and <i>ks_name</i> fields; this triplet uniquely identifies a kstat. If <i>ks_module</i> is <code>NULL</code>, <i>ks_instance</i> is <code>-1</code>, or <i>ks_name</i> is <code>NULL</code>, then those fields will be ignored in the search. For example, <code>kstat_lookup(NULL, -1, "foo")</code> will simply find the first kstat with name "foo".</p> <p>kstat_data_lookup() searches the kstat's data section for the record with the specified <i>name</i>. This operation is only valid for kstat types which have named data records. Currently, only the <code>KSTAT_TYPE_NAMED</code> and <code>KSTAT_TYPE_TIMER</code> kstats have named data records.</p> |
| RETURN VALUES | <p>kstat_lookup() returns a pointer to the requested kstat if it is found, or <code>NULL</code> if it is not.</p> <p>kstat_data_lookup() returns a pointer to the requested data record if it is found. If the requested record is not found, or if the kstat type is invalid, kstat_data_lookup() returns <code>NULL</code>.</p> |
| FILES | <code>/dev/kstat</code> kernel statistics driver |
| SEE ALSO | <code>kstat(3K)</code> , <code>kstat_chain_update(3K)</code> , <code>kstat_close(3K)</code> , <code>kstat_open(3K)</code> , <code>kstat_read(3K)</code> , <code>kstat_write(3K)</code> |

| | |
|----------------------|--|
| NAME | kstat_open, kstat_close – initialize kernel statistics facility |
| SYNOPSIS | <pre> cc [flag ...] file ... -lkstat [library ...] #include <kstat.h> kstat_ctl_t * kstat_open(void); int kstat_close(kstat_ctl_t * kc); </pre> |
| DESCRIPTION | <p>kstat_open() initializes a kstat control structure, which provides access to the kernel statistics library. It returns a pointer to this structure, which must be supplied as the <i>kc</i> argument in subsequent <code>libkstat</code> function calls.</p> <p>kstat_close() frees all resources that were associated with <i>kc</i>. This is done automatically on <code>exit(2)</code> and <code>execve()</code> (see <code>exec(2)</code>).</p> |
| RETURN VALUES | <p>kstat_open() returns a pointer to a kstat control structure. On failure, it returns <code>NULL</code> and no resources are allocated.</p> <p>kstat_close() returns 0 on success, -1 on failure.</p> |
| FILES | <pre> /dev/kstat kernel statistics driver </pre> |
| SEE ALSO | <pre> kstat(3K), kstat_chain_update(3K), kstat_data_lookup(3K), kstat_lookup(3K), kstat_read(3K), kstat_write(3K) </pre> |

| | |
|----------------------|---|
| NAME | kstat_read, kstat_write – read or write kstat data |
| SYNOPSIS | <pre>cc [flag ...] file ... -lkstat [library ...] #include <kstat.h> kid_t kstat_read(kstat_ctl_t * kc, kstat_t * ksp, void * buf); kid_t kstat_write(kstat_ctl_t * kc, kstat_t * ksp, void * buf);</pre> |
| DESCRIPTION | <p>kstat_read() gets data from the kernel for the kstat pointed to by <i>ksp</i>. <i>ksp->ks_data</i> is automatically allocated (or reallocated) to be large enough to hold all of the data. <i>ksp->ks_ndata</i> is set to the number of data fields, <i>ksp->ks_data_size</i> is set to the total size of the data, and <i>ksp->ks_snaptime</i> is set to the high-resolution time at which the data snapshot was taken. If <i>buf</i> is non-NULL, the data is copied from <i>ksp->ks_data</i> into <i>buf</i>.</p> <p>kstat_write() writes data from <i>buf</i>, or from <i>ksp->ks_data</i> if <i>buf</i> is NULL, to the corresponding kstat in the kernel. Only the superuser can use kstat_write().</p> |
| RETURN VALUES | On success, kstat_read() and kstat_write() return the current kstat chain ID (KCID). On failure, they return -1. |
| FILES | /dev/kstat kernel statistics driver |
| SEE ALSO | kstat(3K) , kstat_chain_update(3K) , kstat_close(3K) , kstat_data_lookup(3K) , kstat_lookup(3K) , kstat_open(3K) |

| | |
|----------------------|---|
| NAME | kvm_getu, kvm_getcmd – get the u-area or invocation arguments for a process |
| SYNOPSIS | <pre>#include <kvm.h> #include <sys/param.h> #include <sys/user.h> #include <sys/proc.h> struct user * kvm_getu(kvm_t * <i>kd</i>, struct proc * <i>proc</i>); int kvm_getcmd(kvm_t * <i>kd</i>, struct proc * <i>proc</i>, struct user * <i>u</i>, char *** <i>arg</i>, char *** <i>env</i>);</pre> |
| DESCRIPTION | |
| kvm_getu() | <p>The kvm_getu() function reads the u-area of the process specified by <i>proc</i> to an area of static storage associated with <i>kd</i> and returns a pointer to it. Subsequent calls to kvm_getu() will overwrite this static area.</p> <p>The <i>kd</i> argument is a pointer to a kernel descriptor returned by kvm_open(3K). The <i>proc</i> argument is a pointer to a copy in the current process' address space of a <i>proc</i> structure, obtained, for instance, by a prior kvm_nextproc(3K) call.</p> |
| kvm_getcmd() | <p>The kvm_getcmd() function constructs a list of string pointers that represent the command arguments and environment that were used to initiate the process specified by <i>proc</i>.</p> <p>The <i>kd</i> argument is a pointer to a kernel descriptor returned by kvm_open(3K). The <i>u</i> argument is a pointer to a copy in the current process' address space of a <i>user</i> structure, obtained, for instance, by a prior kvm_getu() call. If <i>arg</i> is not <code>NULL</code>, the command line arguments are formed into a null-terminated array of string pointers. The address of the first such pointer is returned in <i>arg</i>. If <i>env</i> is not <code>NULL</code>, then the environment is formed into a null-terminated array of string pointers. The address of the first of these is returned in <i>env</i>.</p> <p>The pointers returned in <i>arg</i> and <i>env</i> refer to data allocated by malloc(3C) and should be freed by a call to free() when no longer needed. See malloc(3C) Both the string pointers and the strings themselves are deallocated when freed.</p> <p>Since the environment and command line arguments may have been modified by the user process, there is no guarantee that it will be possible to reconstruct the original command at all. Thus, kvm_getcmd() will make the best attempt possible, returning <code>-1</code> if the user process data is unrecognizable.</p> |
| RETURN VALUES | On success, kvm_getu() returns a pointer to a copy of the u-area of the process specified by <i>proc</i> . On failure, it returns <code>NULL</code> . |

The **kvm_getcmd()** function returns 0 on success and -1 on failure.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

kvm_nextproc(3K), **kvm_open(3K)**, **kvm_read(3K)**, **malloc(3C)**, **libkvm(4)**, **attributes(5)**

NOTES

If **kvm_getcmd()** returns -1, the caller still has the option of using the command line fragment that is stored in the u-area.

On systems that support both 32-bit and 64-bit processes, the 64-bit implementation of **libkvm** ensures that the *arg* and *env* pointer arrays for **kvm_getcmd()** are translated to the same form as if they were 64-bit processes. Applications that wish to access the raw 32-bit stack directly can use **kvm_uread()**. See **kvm_read(3K)**.

| | |
|-----------------------|--|
| NAME | kvm_nextproc, kvm_getproc, kvm_setproc – read system process structures |
| SYNOPSIS | <pre>#include <kvm.h> #include <sys/param.h> #include <sys/time.h> #include <sys/proc.h> struct proc * kvm_nextproc(kvm_t * <i>kd</i>); int kvm_setproc(kvm_t * <i>kd</i>); struct proc * kvm_getproc(kvm_t * <i>kd</i>, pid_t <i>pid</i>);</pre> |
| DESCRIPTION | |
| kvm_nextproc() | <p>The kvm_nextproc() function may be used to sequentially read all of the system process structures from the kernel identified by <i>kd</i> (see kvm_open(3K)). Each call to kvm_nextproc() returns a pointer to the static memory area that contains a copy of the next valid process table entry. There is no guarantee that the data will remain valid across calls to kvm_nextproc(), kvm_setproc(), or kvm_getproc(). Therefore, if the process structure must be saved, it should be copied to non-volatile storage.</p> <p>For performance reasons, many implementations will cache a set of system process structures. Since the system state is liable to change between calls to kvm_nextproc(), and since the cache may contain obsolete information, there is no guarantee that every process structure returned refers to an active process, nor is it certain that all processes will be reported.</p> |
| kvm_setproc() | <p>The kvm_setproc() function rewinds the process list, enabling kvm_nextproc() to rescan from the beginning of the system process table. This function will always flush the process structure cache, allowing an application to re-scan the process table of a running system.</p> |
| kvm_getproc() | <p>The kvm_getproc() function locates the <code>proc</code> structure of the process specified by <i>pid</i> and returns a pointer to it. This function does not interact with the process table pointer manipulated by kvm_nextproc(); however, the restrictions regarding the validity of the data still apply.</p> |
| RETURN VALUES | <p>On success, kvm_nextproc() returns a pointer to a copy of the next valid process table entry. On failure, it returns <code>NULL</code>.</p> <p>On success, kvm_getproc() returns a pointer to the <code>proc</code> structure of the process specified by <i>pid</i>. On failure, it returns <code>NULL</code>.</p> |

The **kvm_setproc()** function returns 0 on success -1 on failure.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

kvm_getu(3K) , **kvm_open(3K)** , **kvm_read(3K)** , **attributes(5)**

| NAME | kvm_nlist – get entries from kernel symbol table | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>#include <kvm.h> #include <nlist.h> int kvm_nlist(kvm_t *kd, struct nlist *nl);</pre> | | | | |
| DESCRIPTION | <p>kvm_nlist() examines the symbol table from the kernel image identified by <i>kd</i> (see kvm_open(3K)) and selectively extracts a list of values and puts them in the array of <i>nlist</i> structures pointed to by <i>nl</i>. The name list pointed to by <i>nl</i> consists of an array of structures containing names, types and values. The <i>n_name</i> field of each such structure is taken to be a pointer to a character string representing a symbol name. The list is terminated by an entry with a NULL pointer (or a pointer to a null string) in the <i>n_name</i> field. For each entry in <i>nl</i>, if the named symbol is present in the kernel symbol table, its value and type are placed in the <i>n_value</i> and <i>n_type</i> fields. If a symbol cannot be located, the corresponding <i>n_type</i> field of <i>nl</i> is set to zero.</p> | | | | |
| RETURN VALUES | kvm_nlist() returns the value of nlist(3B) or nlist(3E) , depending on the library used. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | nlist(3B) , nlist(3E) , kvm_open(3K) , kvm_read(3K) , attributes(5) | | | | |

| | | | | | |
|-----------------------|--|-----------------------|------------------|---------------------|------------------------------|
| NAME | kvm_open, kvm_close – specify a kernel to examine | | | | |
| SYNOPSIS | <pre>#include <kvm.h> #include <fcntl.h> kvm_t *kvm_open(char *namelist, char *corefile, char *swapfile, int flag, char *errstr); int kvm_close(kvm_t *kd);</pre> | | | | |
| DESCRIPTION | | | | | |
| kvm_open() | <p>The kvm_open() function initializes a set of file descriptors to be used in subsequent calls to kernel virtual memory (VM) routines. It returns a pointer to a kernel identifier that must be used as the <i>kd</i> argument in subsequent kernel VM function calls.</p> <p>The <i>namelist</i> argument specifies an unstripped executable file whose symbol table will be used to locate various offsets in <i>corefile</i> . If <i>namelist</i> is <code>NULL</code> , the symbol table of the currently running kernel is used to determine offsets in the core image. In this case, it is up to the implementation to select an appropriate way to resolve symbolic references, for instance, using <code>/dev/ksyms</code> as a default <i>namelist</i> file.</p> <p>The <i>corefile</i> argument specifies a file that contains an image of physical memory, for instance, a kernel crash dump file (see savecore(1M)) or the special device <code>/dev/mem</code> . If <i>corefile</i> is <code>NULL</code>, the currently running kernel is accessed, using <code>/dev/mem</code> and <code>/dev/kmem</code> .</p> <p>The <i>swapfile</i> argument specifies a file that represents the swap device. If both <i>corefile</i> and <i>swapfile</i> are <code>NULL</code> , the swap device of the currently running kernel is accessed. Otherwise, if <i>swapfile</i> is <code>NULL</code> , kvm_open() may succeed but subsequent kvm_getu(3K) function calls may fail if the desired information is swapped out.</p> <p>The <i>flag</i> function is used to specify read or write access for <i>corefile</i> and may have one of the following values:</p> <table border="0" style="margin-left: 20px;"> <tr> <td><code>O_RDONLY</code></td> <td>open for reading</td> </tr> <tr> <td><code>O_RDWR</code></td> <td>open for reading and writing</td> </tr> </table> <p>The <i>errstr</i> argument is used to control error reporting. If it is a null pointer, no error messages will be printed. If it is non-null, it is assumed to be the address of a string that will be used to prefix error messages generated by <code>kvm_open</code> . Errors are printed to <code>stderr</code> . A useful value to supply for <i>errstr</i> would be <code>argv [0]</code>. This has the effect of printing the process name in front of any error messages.</p> | <code>O_RDONLY</code> | open for reading | <code>O_RDWR</code> | open for reading and writing |
| <code>O_RDONLY</code> | open for reading | | | | |
| <code>O_RDWR</code> | open for reading and writing | | | | |

Applications using `libkvm` are dependent on the underlying data model of the kernel image, that is, whether it is a 32-bit or 64-bit kernel.

The data model of these applications must match the data model of the kernel in order to correctly interpret the size and offsets of kernel data structures. For example, a 32-bit application that uses the 32-bit version of the `libkvm` interfaces will fail to open a 64-bit kernel image. Similarly, a 64-bit application that uses the 64-bit version of the `libkvm` interfaces will fail to open a 32-bit kernel image.

kvm_close() The `kvm_close()` function closes all file descriptors that were associated with `kd`. These files are also closed on `exit(2)` and `execve()` (see `exec(2)`). `kvm_close()` also resets the `proc` pointer associated with `kvm_nextproc(3K)` and flushes any cached kernel data.

RETURN VALUES

The `kvm_open()` function returns a non-null value suitable for use with subsequent kernel VM function calls. On failure, it returns `NULL` and no files are opened.

The `kvm_close()` function returns 0 on success -1 on failure.

FILES

`/dev/kmem`

`/dev/ksyms`

`/dev/mem`

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`savecore(1M)`, `exec(2)`, `exit(2)`, `pathconf(2)`, `getloadavg(3C)`, `kstat(3K)`, `kvm_getu(3K)`, `kvm_nextproc(3K)`, `kvm_nlist(3K)`, `kvm_read(3K)`, `sysconf(3C)`, `libkvm(4)`, `proc(4)`, `attributes(5)`

NOTES

Kernel core dumps should be examined on the platform on which they were created. While a 32-bit application running on a 64-bit kernel can examine a 32-bit core dump, a 64-bit application running on a 64-bit kernel cannot examine a kernel core dump from the 32-bit system.

Applications using `libkvm` are likely to be platform- and release-dependent.

On 32-bit systems, applications that use `libkvm` to access the running kernel must be 32-bit applications. On systems that support both 32-bit and 64-bit

applications, applications that use the `libkvm` interfaces to access the running kernel must themselves be 64-bit applications.

Most of the traditional uses of `libkvm` have been superseded by more stable interfaces that allow the same information to be extracted more efficiently, yet independent of the kernel data model. For examples, see `sysconf(3C)`, `proc(4)`, `kstat(3K)`, `getloadavg(3C)`, and `pathconf(2)`.

| | |
|---------------------|---|
| NAME | kvm_read, kvm_write, kvm_uread, kvm_uwrite, kvm_kread, kvm_kwrite – copy data to or from a kernel image or running system |
| SYNOPSIS | <pre>#include <kvm.h> ssize_t kvm_read(kvm_t * kd, uintptr_t addr, void * buf, size_t nbytes); ssize_t kvm_write(kvm_t * kd, uintptr_t addr, void * buf, size_t nbytes); ssize_t kvm_kread(kvm_t * kd, uintptr_t addr, void * buf, size_t nbytes); ssize_t kvm_kwrite(kvm_t * kd, uintptr_t addr, void * buf, size_t nbytes); ssize_t kvm_uread(kvm_t * kd, uintptr_t addr, void * buf, size_t nbytes); ssize_t kvm_uwrite(kvm_t * kd, uintptr_t addr, void * buf, size_t nbytes);</pre> |
| DESCRIPTION | |
| kvm_kread() | The kvm_kread() function transfers data from the kernel address space to the address space of the process. <i>nbytes</i> bytes of data are copied from the kernel virtual address given by <i>addr</i> to the buffer pointed to by <i>buf</i> . |
| kvm_kwrite() | The kvm_kwrite() function is like kvm_kread() , except that the direction of the transfer is reversed. To use this function, the kvm_open(3K) call that returned <i>kd</i> must have specified write access. |
| kvm_uread() | The kvm_uread() function transfers data from the address space of the processes specified in the most recent kvm_getu(3K) call. <i>nbytes</i> bytes of data are copied from the user virtual address given by <i>addr</i> to the buffer pointed to by <i>buf</i> . |
| kvm_uwrite() | The kvm_uwrite() function is like kvm_uread() , except that the direction of the transfer is reversed. To use this function, the kvm_open(3K) call that returned <i>kd</i> must have specified write access. The address is resolved in the address space of the process specified in the most recent kvm_getu(3K) call. |
| kvm_read() | The kvm_read() function transfers data from the kernel image specified by <i>kd</i> (see kvm_open(3K)) to the address space of the process. <i>nbytes</i> bytes of data are copied from the kernel virtual address given by <i>addr</i> to the buffer pointed to by <i>buf</i> . |
| kvm_write() | The kvm_write() function is like kvm_read() , except that the direction of data transfer is reversed. To use this function, the kvm_open(3K) call that returned |

kd must have specified write access. If a user virtual address is given, it is resolved in the address space of the process specified in the most recent `kvm_getu(3K)` call.

USAGE

The use of `kvm_read()` and `kvm_write()` is strongly discouraged. On some platforms, there is considerable ambiguity over which address space is to be accessed by these functions, possibly leading to unexpected results. The `kvm_kread()`, `kvm_kwrite()`, `kvm_uread()`, and `kvm_uwrite()` functions are much more clearly defined in this respect.

RETURN VALUES

On success, these functions return the number of bytes actually transferred. On failure, they return `-1`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`kvm_getu(3K)`, `kvm_nlist(3K)`, `kvm_open(3K)`, `attributes(5)`

| NAME | lckpwnf, ulckpwnf – manipulate shadow password database lock file | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <shadow.h> int lckpwnf(void); int ulckpwnf(void);</pre> | | | | |
| DESCRIPTION | <p>The lckpwnf() and ulckpwnf() functions enable modification access to the password databases through the lock file. A process first uses lckpwnf() to lock the lock file, thereby gaining exclusive rights to modify the <code>/etc/passwd</code> or <code>/etc/shadow</code> password database. See passwd(4) and shadow(4). Upon completing modifications, a process should release the lock on the lock file using ulckpwnf(). This mechanism prevents simultaneous modification of the password databases. The lock file, <code>/etc/.pwd.lock</code>, is used to coordinate modification access to the password databases <code>/etc/passwd</code> and <code>/etc/shadow</code>.</p> | | | | |
| RETURN VALUES | <p>If lckpwnf() is successful in locking the file within 15 seconds, it returns 0. If unsuccessful (for example, <code>/etc/.pwd.lock</code> is already locked), it returns -1.</p> <p>If ulckpwnf() is successful in unlocking the file <code>/etc/.pwd.lock</code>, it returns 0. If unsuccessful (for example, <code>/etc/.pwd.lock</code> is already unlocked), it returns -1.</p> | | | | |
| USAGE | These routines are for internal use only; compatibility is not guaranteed. | | | | |
| FILES | <pre>/etc/passwd password database /etc/shadow shadow password database /etc/.pwd.lock lock file</pre> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | getpwnam(3C) , getspnam(3C) , passwd(4) , shadow(4) , attributes(5) | | | | |

| | |
|---------------------------|---|
| NAME | ldap – Lightweight Directory Access Protocol package |
| SYNOPSIS | <pre>cc[<i>flag...</i>] <i>file...</i> -lldap[<i>library...</i>] #include <lber.h> #include <ldap.h></pre> |
| DESCRIPTION | <p>The Lightweight Directory Access Protocol provides TCP/IP access to the X.500 Directory or to a stand-alone LDAP server. The SUNWlldap package includes various LDAP clients and an LDAP client library used to provide programmatic access to the LDAP protocol. This man page gives an overview of the LDAP library functions.</p> <p>Both synchronous and asynchronous APIs are provided. Also included are various functions to parse the results returned from these functions. These functions are found in the <code>libldap.so.3</code> shared object.</p> <p>The basic interaction is as follows. A connection is made to an LDAP server by calling <code>ldap_open(3N)</code>. An LDAP bind operation is performed by calling one of <code>ldap_bind(3N)</code> and friends. Next, other operations are performed by calling one of the synchronous or asynchronous functions (for example, <code>ldap_search_s(3N)</code> or <code>ldap_search(3N)</code> followed by <code>ldap_result(3N)</code>). Results returned from these functions are interpreted by calling the LDAP parsing functions. The LDAP association is terminated by calling <code>ldap_unbind(3N)</code>. Errors can be interpreted by calling <code>ldap_perror(3N)</code>. The <code>ldap_set_rebind_proc(3N)</code> function can be used to set a function to be called back when an LDAP bind operation needs to occur when handling a client referral.</p> |
| Search Filters | Search filters to be passed to the ldap search functions can be constructed by hand, or by calling the <code>ldap_getfilter(3N)</code> functions. |
| Displaying Results | <p>Results obtained from the ldap search functions can be output by hand, by calling <code>ldap_first_entry(3N)</code> and <code>ldap_next_entry(3N)</code> to step through the entries returned, <code>ldap_first_attribute(3N)</code> and <code>ldap_next_attribute(3N)</code> to step through an entry's attributes, and <code>ldap_get_values(3N)</code> to retrieve a given attribute's value, and then calling <code>printf(3S)</code> or whatever to display the values.</p> <p>Alternatively, the entry can be output automatically by calling the <code>ldap_entry2text(3N)</code>, <code>ldap_entry2text_search(3N)</code>, <code>ldap_entry2html(3N)</code>, or <code>ldap_entry2html_search(3N)</code> functions. These functions look up the object class of the entry they are passed in the <code>ldaptemplates.conf(4)</code> file to decide which attributes to display and how to display them. Output is handled via a function passed in as a parameter.</p> |

| | | |
|---|---|--|
| Uniform Resource Locators (URLS) | The <code>ldap_ur1(3N)</code> functions can be used test a URL to see if it is an LDAP URL, to parse LDAP URLs into their component pieces, to initiate searches directly using an LDAP URL, and to retrieve the URL associated with a DNS domain name or a distinguished name. | |
| User Friendly Naming | The <code>ldap_ufn(3N)</code> functions implement a user friendly naming scheme via LDAP. This scheme allows you to look up entries using fuzzy, untyped names like "mark smith, umich, us". | |
| Caching | The <code>ldap_cache(3N)</code> functions implement a local client caching scheme, providing a substantial performance increase for repeated queries. | |
| Utility Functions | Also provided are various utility functions. The <code>ldap_sort(3N)</code> functions are used to sort the entries and values returned via the ldap search functions. The <code>ldap_friendly(3N)</code> functions are used to map from short two letter country codes (or other strings) to longer "friendlier" names. The <code>ldap_charset(3N)</code> functions can be used to translate to and from the T.61 character set used for many character strings in the LDAP protocol. | |
| Connectionless Access | The <code>clldap_search_s(3N)</code> function allows you to access the directory via Connectionless LDAP (CLDAP), which is similar to LDAP but operates over UDP, obviating the need to set up and tear down a connection by calling <code>ldap_open(3N)</code> , <code>ldap_bind(3N)</code> , and <code>ldap_unbind(3N)</code> . <code>clldap_open(3N)</code> should be called before using <code>clldap_search_s(3N)</code> . All the same getfilter, parsing, and display that can be used with regular LDAP functions can be used with the CLDAP functions. | |
| BER Library | Also included in the distribution is a set of lightweight Basic Encoding Rules functions. These functions are used by the LDAP library functions to encode and decode LDAP protocol elements using the (slightly simplified) Basic Encoding Rules defined by LDAP. They are not normally used directly by an LDAP application program. The functions provide a printf and scanf-like interface, as well as lower-level access. | |
| Index | <code>ldap_open(3N)</code> | open a connection to an LDAP server |
| | <code>ldap_init(3N)</code> | initialize the LDAP library without opening a connection to a server |
| | <code>ldap_result(3N)</code> | wait for the result from an asynchronous operation |

| | |
|-------------------------------------|---|
| <code>ldap_abandon(3N)</code> | abandon (abort) an asynchronous operation |
| <code>ldap_add(3N)</code> | asynchronously add an entry |
| <code>ldap_add_s(3N)</code> | synchronously add an entry |
| <code>ldap_add_ext(3N)</code> | asynchronously add an entry, return value and place message |
| <code>ldap_add_ext_s(3N)</code> | synchronously add an entry, return value and place message |
| <code>ldap_bind(3N)</code> | asynchronously bind to the directory |
| <code>ldap_bind_s(3N)</code> | synchronously bind to the directory |
| <code>ldap_simple_bind(3N)</code> | asynchronously bind to the directory using simple authentication |
| <code>ldap_simple_bind_s(3N)</code> | synchronously bind to the directory using simple authentication |
| <code>ldap_unbind(3N)</code> | synchronously unbind from the LDAP server and close the connection |
| <code>ldap_unbind_s(3N)</code> | equivalent to <code>ldap_unbind(3N)</code> |
| <code>ldap_memfree(3N)</code> | dispose of memory allocated by LDAP functions (this is only used on the Microsoft Windows platforms; use <code>free(3C)</code> on all other platforms). |
| <code>ldap_enable_cache(3N)</code> | enable LDAP client caching |
| <code>ldap_disable_cache(3N)</code> | disable LDAP client caching |
| <code>ldap_destroy_cache(3N)</code> | disable LDAP client caching and destroy cache contents |

| | |
|--|---|
| <code>ldap_flush_cache(3N)</code> | flush LDAP client cache |
| <code>ldap_uncache_entry(3N)</code> | uncache requests pertaining to an entry |
| <code>ldap_uncache_request(3N)</code> | uncache a request |
| <code>ldap_set_cache_options(3N)</code> | set cache options |
| <code>ldap_compare(3N)</code> | asynchronous compare to a directory entry |
| <code>ldap_compare_s(3N)</code> | synchronous compare to a directory entry |
| <code>ldap_compare_ext(3N)</code> | asynchronous compare to a directory entry, return value and place message |
| <code>ldap_compare_ext_s(3N)</code> | synchronous compare to a directory entry, return value and place message |
| <code>ldap_control_free(3N)</code> | LDAP control disposal |
| <code>ldap_controls_free_(3N)</code> | LDAP control disposal |
| <code>ldap_delete(3N)</code> | asynchronously delete an entry |
| <code>ldap_delete_s(3N)</code> | synchronously delete an entry |
| <code>ldap_delete_ext(3N)</code> | asynchronously delete an entry, return value and place message |
| <code>ldap_delete_ext_s(3N)</code> | synchronously delete an entry, return value and place |
| <code>ldap_init_templates(3N)</code> | initialize display template functions from a file |
| <code>ldap_init_templates_buf(3N)</code> | initialize display template functions from a buffer |
| <code>ldap_free_templates(3N)</code> | free display template function memory |

| | |
|---------------------------------------|--|
| <code>ldap_first_reference(3N)</code> | steps through <code>ldap_result(3N)</code> message chain |
| <code>ldap_count_reference(3N)</code> | counts the messages in an <code>ldap_result(3N)</code> message chain |
| <code>ldap_next_reference(3N)</code> | steps through <code>ldap_result(3N)</code> message chain |
| <code>ldap_first_message(3N)</code> | steps through <code>ldap_result(3N)</code> message chain |
| <code>ldap_count_messages(3N)</code> | counts the messages in an <code>ldap_result(3N)</code> message chain |
| <code>ldap_next_message(3N)</code> | steps through <code>ldap_result(3N)</code> message chain |
| <code>ldap_msgtype(3N)</code> | returns the type of LDAP message |
| <code>ldap_first_disptmpl(3N)</code> | get first display template |
| <code>ldap_next_disptmpl(3N)</code> | get next display template |
| <code>ldap_oc2template(3N)</code> | return template appropriate for objectclass |
| <code>ldap_name2template(3N)</code> | return named template |
| <code>ldap_tmplattrs(3N)</code> | return attributes needed by template |
| <code>ldap_first_tmplrow(3N)</code> | return first row of displayable items in a template |
| <code>ldap_next_tmplrow(3N)</code> | return next row of displayable items in a template |

| | |
|---|---|
| <code>ldap_first_tmplcol(3N)</code> | return first column of displayable items in a template |
| <code>ldap_next_tmplcol(3N)</code> | return next column of displayable items in a template |
| <code>ldap_entry2text(3N)</code> | display an entry as text using a display template |
| <code>ldap_entry2text_search(3N)</code> | search for and display an entry as text using a display template |
| <code>ldap_vals2text(3N)</code> | display values as text |
| <code>ldap_entry2html(3N)</code> | display an entry as HTML (HyperText Markup Language) using a display template |
| <code>ldap_entry2html_search(3N)</code> | search for and display an entry as HTML using a display template |
| <code>ldap_vals2html(3N)</code> | display values as HTML |
| <code>ldap_perror(3N)</code> | print an LDAP error indication to standard error |
| <code>ld_errno(3N)</code> | LDAP error indication |
| <code>ldap_result2error(3N)</code> | extract LDAP error indication from LDAP result |
| <code>ldap_errlist(3N)</code> | list of ldap errors and their meanings |
| <code>ldap_err2string(3N)</code> | convert LDAP error indication to a string |
| <code>ldap_first_attribute(3N)</code> | return first attribute name in an entry |
| <code>ldap_next_attribute(3N)</code> | return next attribute name in an entry |

| | |
|--|--|
| <code>ldap_first_entry(3N)</code> | return first entry in a chain of search results |
| <code>ldap_next_entry(3N)</code> | return next entry in a chain of search results |
| <code>ldap_count_entries(3N)</code> | return number of entries in a search result |
| <code>ldap_friendly_name(3N)</code> | map from unfriendly to friendly names |
| <code>ldap_free_friendlymap(3N)</code> | free resources used by <code>ldap_friendly(3N)</code> |
| <code>ldap_get_dn(3N)</code> | extract the DN from an entry |
| <code>ldap_explode_dn(3N)</code> | convert a DN into its component parts |
| <code>ldap_explode_dns(3N)</code> | convert a DNS-style DN into its component parts (experimental) |
| <code>ldap_is_dns_dn(3N)</code> | check to see if a DN is a DNS-style DN (experimental) |
| <code>ldap_dns_to_dn(3N)</code> | convert a DNS domain name into an X.500 distinguished name |
| <code>ldap_dn2ufn(3N)</code> | convert a DN into user friendly form |
| <code>ldap_get_values(3N)</code> | return an attribute's values |
| <code>ldap_get_values_len(3N)</code> | return an attribute values with lengths |
| <code>ldap_value_free(3N)</code> | free memory allocated by <code>ldap_get_values(3N)</code> |
| <code>ldap_value_free_len(3N)</code> | free memory allocated by <code>ldap_get_values_len(3N)</code> |
| <code>ldap_count_values(3N)</code> | return number of values |

| | |
|--|--|
| <code>ldap_count_values_len(3N)</code> | return number of values |
| <code>ldap_init_getfilter(3N)</code> | initialize getfilter functions from a file |
| <code>ldap_init_getfilter_buf(3N)</code> | initialize getfilter functions from a buffer |
| <code>ldap_getfilter_free(3N)</code> | free resources allocated by <code>ldap_init_getfilter (3N)</code> |
| <code>ldap_getfirstfilter(3N)</code> | return first search filter |
| <code>ldap_getnextfilter(3N)</code> | return next search filter |
| <code>ldap_build_filter(3N)</code> | construct an LDAP search filter from a pattern |
| <code>ldap_setfilteraffixes(3N)</code> | set prefix and suffix for search filters |
| <code>ldap_modify(3N)</code> | asynchronously modify an entry |
| <code>ldap_modify_s(3N)</code> | synchronously modify an entry |
| <code>ldap_modify_ext(3N)</code> | asynchronously modify an entry, return value, place message |
| <code>ldap_modify_ext_s(3N)</code> | synchronously modify an entry, return value, place message |
| <code>ldap_mods_free(3N)</code> | free array of pointers to mod structures used by <code>ldap_modify (3N)</code> |
| <code>ldap_modrdn2(3N)</code> | asynchronously modify the RDN of an entry |
| <code>ldap_modrdn2_s(3N)</code> | synchronously modify the RDN of an entry |
| <code>ldap_modrdn(3N)</code> | deprecated - use <code>ldap_modrdn2 (3N)</code> |

| | |
|---|---|
| <code>ldap_modrdn_s(3N)</code> | deprecated - use <code>ldap_modrdn2_s(3N)</code> |
| <code>ldap_rename(3N)</code> | asynchronously modify the name of an LDAP entry |
| <code>ldap_rename_s(3N)</code> | synchronously modify the name of an LDAP entry |
| <code>ldap_msgfree(3N)</code> | free results allocated by <code>ldap_result(3N)</code> |
| <code>ldap_parse_result(3N)</code> | search for a message to parse |
| <code>ldap_parse_extended_result(3N)</code> | search for a message to parse |
| <code>ldap_parse_saslbind_result(3N)</code> | search for a message to parse |
| <code>ldap_search(3N)</code> | asynchronously search the directory |
| <code>ldap_search_s(3N)</code> | synchronously search the directory |
| <code>ldap_search_ext(3N)</code> | asynchronously search the directory, return value and place message |
| <code>ldap_search_ext_s(3N)</code> | synchronously search the directory, return value and place message |
| <code>ldap_search_st(3N)</code> | synchronously search the directory with timeout |
| <code>ldap_ufn_search_s(3N)</code> | user friendly search the directory |
| <code>ldap_ufn_search_c(3N)</code> | user friendly search the directory with cancel |
| <code>ldap_ufn_search_ct(3N)</code> | user friendly search the directory with cancel and timeout |

| | |
|--|---|
| <code>ldap_ufn_setfilter(3N)</code> | set filter file used by ldap_ufn (3N) functions |
| <code>ldap_ufn_setprefix(3N)</code> | set prefix used by ldap_ufn (3N) functions |
| <code>ldap_ufn_timeout(3N)</code> | set timeout used by ldap_ufn (3N) functions |
| <code>ldap_is_ldap_url(3N)</code> | check a URL string to see if it is an LDAP URL |
| <code>ldap_url_parse(3N)</code> | break up an LDAP URL string into its components |
| <code>ldap_url_search(3N)</code> | asynchronously search using an LDAP URL |
| <code>ldap_url_search_s(3N)</code> | synchronously search using an LDAP URL |
| <code>ldap_url_search_st(3N)</code> | synchronously search using an LDAP URL and a timeout |
| <code>ldap_dns_to_url(3N)</code> | locate the LDAP URL associated with a DNS domain name. |
| <code>ldap_dn_to_url(3N)</code> | locate the LDAP URL associated with a distinguished name. |
| <code>ldap_init_searchprefs(3N)</code> | initialize searchprefs functions from a file |
| <code>ldap_init_searchprefs_buf(3N)</code> | initialize searchprefs functions from a buffer |
| <code>ldap_free_searchprefs(3N)</code> | free memory allocated by searchprefs functions |
| <code>ldap_first_searchobj(3N)</code> | return first searchpref object |
| <code>ldap_next_searchobj(3N)</code> | return next searchpref object |
| <code>ldap_sort_entries(3N)</code> | sort a list of search results |
| <code>ldap_sort_values(3N)</code> | sort a list of attribute values |

| | |
|--|--|
| <code>ldap_sort_strcasecmp(3N)</code> | case insensitive string comparison |
| <code>ldap_set_string_translators(3N)</code> | set character set translation functions used by LDAP library |
| <code>ldap_t61_to_8859(3N)</code> | translate from ISO-8859 characters to the T.61 characters |
| <code>ldap_8859_to_t61(3N)</code> | translate from T.61 characters to the ISO-8859 characters |
| <code>ldap_translate_from_t61(3N)</code> | translate from the T.61 character set to another character set |
| <code>ldap_translate_to_t61(3N)</code> | translate to the T.61 character set from another character set |
| <code>ldap_enable_translation(3N)</code> | enable or disable character translation for an LDAP entry result |
| <code>cldap_open(3N)</code> | open a connectionless LDAP (CLDAP) session |
| <code>cldap_search_s(3N)</code> | perform a search using connectionless LDAP |
| <code>cldap_setretryinfo(3N)</code> | set retry and timeout information using connectionless LDAP |
| <code>cldap_close(3N)</code> | terminate a connectionless LDAP session |

ATTRIBUTES

See `attributes(5)` for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|--|
| Availability | SUNWlldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

| NAME | ldap_abandon – abandon an LDAP operation in progress | | | | | | |
|--------------------|--|----------------|-----------------|--------------|---|-----------------|----------|
| SYNOPSIS | <pre>cc[<i>flag...</i>] <i>file...</i> -ldap[<i>library...</i>] #include <lber.h> #include <ldap.h> int ldap_abandon(LDAP *ld, int msgid);</pre> | | | | | | |
| DESCRIPTION | <p>The ldap_abandon() function is used to abandon or cancel an LDAP operation in progress. The <i>msgid</i> passed should be the message id of an outstanding LDAP operation, as returned by ldap_search(3N), ldap_modify(3N), etc.</p> <p>ldap_abandon() checks to see if the result of the operation has already come in. If it has, it deletes it from the queue of pending messages. If not, it sends an LDAP abandon operation to the the LDAP server.</p> <p>The caller can expect that the result of an abandoned operation will not be returned from a future call to ldap_result(3N).</p> | | | | | | |
| ERRORS | ldap_abandon() returns 0 if successful or -1 otherwise and setting <i>ld_errno</i> appropriately. See ldap_error(3N) for details. | | | | | | |
| ATTRIBUTES | See attributes(5) for a description of the following attributes: | | | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>Availability</td> <td>SUNWldap (32-bit) SUNWldapx (64-bit)</td> </tr> <tr> <td>Stability Level</td> <td>Evolving</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | Availability | SUNWldap (32-bit) SUNWldapx (64-bit) | Stability Level | Evolving |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) | | | | | | |
| Stability Level | Evolving | | | | | | |
| SEE ALSO | ldap(3N), ldap_result(3N), ldap_error(3N) | | | | | | |

| | |
|--------------------|---|
| NAME | ldap_add, ldap_add_s, ldap_add_ext, ldap_add_ext_s – perform an LDAP add operation |
| SYNOPSIS | <pre> cc[flag ...] file ... -lldap[library ...] #include <lber.h> #include <ldap.h> int ldap_add(LDAP *ld, char *dn, LDAPMod *attrs []); int ldap_add_s(LDAP *ld, char *dn, LDAPMod *attrs []); int ldap_add_ext(LDAP *ld, char *dn, LDAPMod **attrs, LDAPControl **serverctrls, int * msgidp); int ldap_add_ext_s(LDAP *ld, char *dn, LDAPMod **attrs, LDAPControl **serverctrls, LDAPControl **clientctrls); </pre> |
| DESCRIPTION | <p>The ldap_add_s() function is used to perform an LDAP add operation. It takes <i>dn</i>, the DN of the entry to add, and <i>attrs</i>, a null-terminated array of the entry's attributes. The LDAPMod structure is used to represent attributes, with the <i>mod_type</i> and <i>mod_values</i> fields being used as described under ldap_modify(3N), and the <i>ldap_op</i> field being used only if you need to specify the LDAP_MOD_BVALUES option. Otherwise, it should be set to zero.</p> <p>Note that all entries except that specified by the last component in the given DN must already exist. ldap_add_s() returns an LDAP error code indicating success or failure of the operation. See ldap_error(3N) for more details.</p> <p>The ldap_add() function works just like ldap_add_s(), but it is asynchronous. It returns the message id of the request it initiated. The result of this operation can be obtained by calling ldap_result(3N).</p> <p>The ldap_add_ext() function initiates an asynchronous add operation and returns LDAP_SUCCESS if the request was successfully sent to the server, or else it returns a LDAP error code if not (see ldap_error(3n)). If successful, ldap_add_ext() places the message id of <i>*msgidp</i>. A subsequent call to ldap_result(), can be used to obtain the result of the add request.</p> <p>The ldap_add_ext_s() function initiates a synchronous add operation and returns the result of the operation itself.</p> |

ERRORS

ldap_add() returns - 1 in case of error initiating the request, and will set the *ld_errno* field in the *ld* parameter to indicate the error. **ldap_add_s()** will return an LDAP error code directly (LDAP_SUCCESS if everything went ok, an error otherwise).

ATTRIBUTES

See **attributes(5)** for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|---|
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

ldap(3N) , **ldap_error(3N)** , **ldap_modify(3N)**

| | |
|------------------------------|---|
| NAME | ldap_bind, ldap_bind_s, ldap_sasl_bind, ldap_sasl_bind_s, ldap_simple_bind, ldap_simple_bind_s, ldap_unbind, ldap_unbind_s, ldap_set_rebind_proc – LDAP bind functions |
| SYNOPSIS | <pre> cc[flag ...] file ... -lldap[library ...] #include <lber.h> #include <ldap.h> int ldap_bind(LDAP *ld, char *who, char *cred, int method); int ldap_bind_s(LDAP *ld, char *who, char *cred, int method); int ldap_simple_bind(LDAP *ld, char *who, char *passwd); int ldap_simple_bind_s(LDAP *ld, char *who, char *passwd); int ldap_unbind(LDAP *ld); int ldap_unbind_s(LDAP *ld); void ldap_set_rebind_proc(LDAP *ld, int (*rebindproc); int ldap_sasl_bind(LDAP *ld, char *dn, char *mechanism, struct berval *cred, LDAPControl **serverctrls, LDAPControl **clientctrls, int *msgidp); int ldap_sasl_bind_s(LDAP *ld, char *dn, char *mechanism, struct berval *cred, LDAPControl **serverctrls, LDAPControl **clientctrls); </pre> |
| DESCRIPTION | <p>These functions provide various interfaces to the LDAP bind operation. After a connection is made to an LDAP server using <code>ldap_open(3N)</code>, an LDAP bind operation must be performed before other operations can be attempted over the connection. Both synchronous and asynchronous versions of each variant of the bind call are provided. There are three types of calls, providing simple authentication, kerberos authentication, and general functions to do either one. All functions take <code>ld</code> as their first parameter, as returned from <code>ldap_open(3N)</code>.</p> |
| Simple Authentication | <p>The simplest form of the bind call is <code>ldap_simple_bind_s()</code>. It takes the DN to bind as in <code>who</code>, and the userPassword associated with the entry in <code>passwd</code>. It returns an LDAP error indication (see <code>ldap_error(3N)</code>). The <code>ldap_simple_bind()</code> call is asynchronous, taking the same parameters but only initiating the bind operation and returning the message id of the request it</p> |

General Authentication

sent. The result of the operation can be obtained by a subsequent call to `ldap_result(3N)`.

The `ldap_bind()` and `ldap_bind_s()` functions can be used when the authentication method to use needs to be selected at runtime. They both take an extra *method* parameter selecting the authentication method to use. It should be set to `LDAP_AUTH_SIMPLE` to select simple authentication. `ldap_bind()` returns the message id of the request it initiates. `ldap_bind_s()` returns an LDAP error indication.

The `ldap_sasl_bind()` and `ldap_sasl_bind_s()` functions are used for general and extensible authentication over LDAP through the use of the Simple Authentication Security Layer. The routines both take the dn to bind as, the method to use, as a dotted-string representation of an OID identifying the method, and a struct `berval` holding the credentials. The special constant value `LDAP_SASL_SIMPLE` ("") can be passed to request simple authentication, or the simplified routines `ldap_simple_bind()` or `ldap_simple_bind_s()` can be use.

Unbinding

The `ldap_unbind()` call is used to unbind from the directory, terminate the current association, and free the resources contained in the `ld` structure. Once it is called, the connection to the LDAP server is closed, and the `ld` structure is invalid. The `ldap_unbind_s()` call is just another name for `ldap_unbind()`; both of these calls are synchronous in nature.

Re-Binding While Following Referral

The `ldap_set_rebind_proc()` call is used to set a function that will be called back to obtain bind credentials used when a new server is contacted during the following of an LDAP referral. Note that this function is only available when the LDAP libraries are compiled with `LDAP_REFERRALS` defined and is only used when the `ld_options` field in the LDAP structure has `LDAP_OPT_REFERRALS` set (this is the default). If `ldap_set_rebind_proc()` is never called, or if it is called with a NULL *rebindproc* parameter, an unauthenticated simple LDAP bind will always be done when chasing referrals.

rebindproc should be a function that is declared like this:

```
int rebindproc( LDAP *ld, char **whop, char **credp,
               int *methodp, int freeit );
```

The LDAP library will first call the `rebindproc` to obtain the referral bind credentials, and the *freeit* parameter will be zero. The *whop*, *credp*, and *methodp* should be set as appropriate. If the `rebindproc` returns `LDAP_SUCCESS`, referral processing continues, and the `rebindproc` will be called a second time with *freeit* non-zero to give your application a chance to free any memory allocated in the previous call.

If anything but `LDAP_SUCCESS` is returned by the first call to the `rebindproc`, then referral processing is stopped and that error code is returned for the original LDAP operation.

RETURN VALUES

A call to `ldap_result(3N)`, can be used to obtain the result of the bind operations.

ERRORS

Asynchronous functions will return `- 1` in case of error, setting the `ld_errno` parameter of the `ld` structure. Synchronous functions return whatever `ld_errno` is set to. See `ldap_error(3N)` for more information. If no credentials are returned the result parameter is set to `NULL`.

ATTRIBUTES

See `attributes(5)` for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|---|
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

`ldap(3N)`, `ldap_error(3N)`, `ldap_open(3N)`

NAME ldap_cache, ldap_enable_cache, ldap_disable_cache, ldap_destroy_cache, ldap_flush_cache, ldap_uncache_entry, ldap_uncache_request, ldap_set_cache_options – LDAP client caching functions

SYNOPSIS

```
cc[
  flag
  ... ]
file
... -lldap[
  library
  ... ]

#include <lber.h>
#include <ldap.h>

ldap_enable_cache(LDAP *ld, long timeout, long maxmem);

void ldap_disable_cache(LDAP *ld);

void ldap_destroy_cache(LDAP *ld);

void ldap_flush_cache(LDAP *ld);

void ldap_uncache_entry(LDAP *ld, char *dn);

void ldap_uncache_request(LDAP *ld, int msgid);

void ldap_set_cache_options(LDAP *ld, unsigned long opts);
```

DESCRIPTION

These functions are used to control the behavior of client caching of `ldap_search(3N)`, `clldap_search_s(3N)`, and `ldap_compare(3N)` operations. By default, the cache is disabled and no caching is done. Enabling the cache can greatly improve performance and reduce network bandwidth when a client DUA makes repeated requests.

`ldap_enable_cache()` should be called to turn on local caching or to change cache parameters (lifetime of cached requests and memory used). The `ld` parameter should be the result of a successful call to `ldap_open(3N)`. The `timeout` is specified in seconds, and is used to decide how long to keep cached requests. The `maxmem` value is in bytes, and is used to set an upper bound on how memory the cache will use. You can specify 0 for `maxmem` to restrict the cache size by the `timeout` only. The first call to `ldap_enable_cache` creates the cache; subsequent calls re-enable the cache and set the timeout and memory values.

`ldap_disable_cache()` temporarily disables use of the cache (new requests are not cached and the cache is not checked when returning results). It does not delete the cache contents.

ldap_destroy_cache() turns off caching and completely removes the cache from memory.

ldap_flush_cache() deletes the cache contents, but does not effect it in any other way.

ldap_uncache_entry() removes all requests that make reference to the distinguished name *dn* from the cache. It should be used, for example, after doing an **ldap_modify(3N)** call involving *dn* .

ldap_uncache_request() removes the request indicated by the LDAP request id *msgid* from the cache.

ldap_set_cache_options() is used to change caching behavior. The current supported options are `LDAP_CACHE_OPT_CACHENOERRS` to suppress caching of any requests that result in an error, and `LDAP_CACHE_OPT_CACHEALLERRS` to enable caching of all requests. The default behavior is to not cache requests that result in errors, except that request that result in the error `LDAP_SIZELIMIT_EXCEEDED` are cached.

ERRORS

ldap_enable_cache() returns 0 upon success, and - 1 if it is unable to allocate space for the cache. All the other calls are declared as void and return nothing.

ATTRIBUTES

See **attributes(5)** for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|--|
| Availability | SUNWlldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

ldap(3N) , **ldap_search(3N)** , **ldap_compare(3N)** ,
cldap_search_s(3N)

NAME ldap_charset, ldap_set_string_translators, ldap_t61_to_8859, ldap_8859_to_t61, ldap_translate_from_t61, ldap_translate_to_t61, ldap_enable_translation – LDAP character set translation functions

SYNOPSIS

```
cc[
flag
... ]
file
... -lldap[
library
... ]

#include <lber.h>
#include <ldap.h>

void ldap_set_string_translators(LDAP *ld, BERTranslateProc encode_proc,
BERTranslateProc decodeproc);

typedef int(*BERTranslateProc)(char **bufp, unsigned long *buflenp, int free_input);

int ldap_t61_to_8859(char **bufp, unsigned long *buflenp, int free_input);

int ldap_8859_to_t61(char **bufp, unsigned long *buflenp, int free_input);

int ldap_translate_from_t61(LDAP *ld, char **bufp, unsigned long *lenp, int
free_input);

int ldap_translate_to_t61(LDAP *ld, char **bufp, unsigned long *lenp, int
free_input);

void ldap_enable_translation(LDAP *ld, LDAPMessage *entry, int enable);
```

DESCRIPTION

These functions are used to enable translation of character strings used in the LDAP library to and from the T.61 character set used in the LDAP protocol. These functions are only available if the LDAP and LBER libraries are compiled with `STR_TRANSLATION` defined. It is also possible to turn on character translation by default so that all LDAP library callers will experience translation; see the LDAP Make-common source file for details.

ldap_set_string_translators() sets the translation functions that will be used by the LDAP library. They are not actually used until the `ld_lberoptions` field of the LDAP structure is set to include the `LBER_TRANSLATE_STRINGS` option.

ldap_t61_to_8859() and **ldap_8859_to_t61()** are translation functions for converting between T.61 characters and ISO-8859 characters. The specific 8859 character set used is determined at compile time.

ldap_translate_from_t61() is used to translate a string of characters from the T.61 character set to a different character set. The actual translation is done

using the *decode_proc* that was passed to a previous call to `ldap_set_string_translators()`. On entry, **bufp* should point to the start of the T.61 characters to be translated and **lenp* should contain the number of bytes to translate. If *free_input* is non-zero, the input buffer will be freed if translation is a success. If the translation is a success, `LDAP_SUCCESS` will be returned, **bufp* will point to a newly malloc'd buffer that contains the translated characters, and **lenp* will contain the length of the result. If translation fails, an LDAP error code will be returned.

ldap_translate_to_t61() is used to translate a string of characters to the T.61 character set from a different character set. The actual translation is done using the *encode_proc* that was passed to a previous call to `ldap_set_string_translators()`. This function is called just like `ldap_translate_from_t61()`.

ldap_enable_translation() is used to turn on or off string translation for the LDAP entry *entry* (typically obtained by calling **ldap_first_entry()** or **ldap_next_entry()** after a successful LDAP search operation). If *enable* is zero, translation is disabled; if non-zero, translation is enabled. This function is useful if you need to ensure that a particular attribute is not translated when it is extracted using **ldap_get_values()** or **ldap_get_values_len()**. For example, you would not want to translate a binary attributes such as `jpegPhoto`.

ATTRIBUTES

See **attributes(5)** for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|--|
| Availability | SUNWlldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

ldap(3N)

| | |
|--------------------|---|
| NAME | ldap_compare, ldap_compare_s, ldap_compare_ext, ldap_compare_ext_s – LDAP compare operation |
| SYNOPSIS | <pre>cc[flag ...] file ... -lldap[library ...] #include <lber.h> #include <ldap.h> int ldap_compare(LDAP *ld, char *dn, char *attr, char *value); int ldap_compare_s(LDAP *ld, char *dn, char *attr, char *value); int ldap_compare_ext(LDAP *ld, char *dn, char *attr, struct berval *bvalue, LDAPControl **serverctrls, LDAPControl **clientctrls, int *msgidp); int ldap_compare_ext_s(LDAP *ld, char *dn, char *attr, struct berval *bvalue, LDAPControl **serverctrls, LDAPControl **clientctrls);</pre> |
| DESCRIPTION | <p>The ldap_compare_s() function is used to perform an LDAP compare operation synchronously. It takes <i>dn</i>, the DN of the entry upon which to perform the compare, and <i>attr</i> and <i>value</i>, the attribute type and value to compare to those found in the entry. It returns an LDAP error code, which will be <code>LDAP_COMPARE_TRUE</code> if the entry contains the attribute value and <code>LDAP_COMPARE_FALSE</code> if it does not. Otherwise, some error code is returned.</p> <p>The ldap_compare() function is used to perform an LDAP compare operation asynchronously. It takes the same parameters as ldap_compare_s(), but returns the message id of the request it initiated. The result of the compare can be obtained by a subsequent call to ldap_result(3N).</p> <p>The ldap_compare_ext() function initiates an asynchronous compare operation and returns <code>LDAP_SUCCESS</code> if the request was successfully sent to the server, or else it returns a LDAP error code if not (see ldap_error(3n)). If successful, ldap_compare_ext() places the message id of the request in <i>*msgidp</i>. A subsequent call to ldap_result(), can be used to obtain the result of the add request.</p> <p>The ldap_compare_ext_s() function initiates a synchronous compare operation and as such returns the result of the operation itself.</p> |
| ERRORS | ldap_compare_s() returns an LDAP error code which can be interpreted by calling one of ldap_perror(3N) and friends. ldap_compare() returns <code>- 1</code> if |

something went wrong initiating the request. It returns the non-negative message id of the request if it was successful.

ATTRIBUTES

See **attributes(5)** for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|---|
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

ldap(3N) , **ldap_error(3N)**

BUGS

There is no way to compare binary values but there should be.

NAME ldap_control_free, ldap_controls_free – LDAP control disposal

SYNOPSIS

```
cc[
  flag
  ... ]
file
... -lldap[
  library
  ... ]

#include <lber.h>
#include <ldap.h>

void ldap_control_free(LDAPControl *ctrl);
void ldap_controls_free(LDAPControl *ctrls);
```

DESCRIPTION ldap_controls_free() and ldap_control_free() are routines which can be used to dispose of a single control or an array of controls allocated by other LDAP APIs.

RETURN VALUES None.

ERRORS No errors are defined for these functions.

ATTRIBUTES See attributes(5) for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|---|
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO ldap_error(3N), ldap_result(3N), attributes(5)

| | |
|--------------------|--|
| NAME | ldap_delete, ldap_delete_s, ldap_delete_ext, ldap_delete_ext_s – LDAP delete operation |
| SYNOPSIS | <pre> cc[flag ...] file ... -lldap[library ...] #include <lber.h> #include <ldap.h> int ldap_delete(LDAP *ld, char *dn); int ldap_delete_s(LDAP *ld, char *dn); int ldap_delete_ext(LDAP *ld, char *dn, LDAPControl **serverctrls, LDAPControl **clientctrls, int *msgidp); int ldap_delete_ext_s(LDAP *ld, char *dn, LDAPControl **serverctrls, LDAPControl **clientctrls); </pre> |
| DESCRIPTION | <p>The ldap_delete_s() function is used to perform an LDAP delete operation synchronously. It takes <i>dn</i>, the DN of the entry to be deleted. It returns an LDAP error code, indicating the success or failure of the operation.</p> <p>The ldap_delete() function is used to perform an LDAP delete operation asynchronously. It takes the same parameters as ldap_delete_s(), but returns the message id of the request it initiated. The result of the delete can be obtained by a subsequent call to ldap_result(3N).</p> <p>The ldap_delete_ext() function initiates an asynchronous delete operation and returns <code>LDAP_SUCCESS</code> if the request was successfully sent to the server, or else it returns a LDAP error code if not (see ldap_error(3N)). If successful, ldap_delete_ext() places the message id of the request in <i>*msgidp</i>. A subsequent call to ldap_result(), can be used to obtain the result of the add request.</p> <p>The ldap_delete_ext_s() function initiates a synchronous delete operation and as such returns the result of the operation itself.</p> |
| ERRORS | <p>ldap_delete_s() returns an LDAP error code which can be interpreted by calling one of ldap_perror(3N) functions. ldap_delete() returns <code>- 1</code> if something went wrong initiating the request. It returns the non-negative message id of the request if things were successful.</p> |

ATTRIBUTES

See **attributes(5)** for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|---|
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

ldap(3N) , **ldap_error(3N)**

| | |
|--------------------|--|
| NAME | ldap_disptmpl, ldap_init_templates, ldap_init_templates_buf, ldap_free_templates, ldap_first_disptmpl, ldap_next_disptmpl, ldap_oc2template, ldap_tmplattrs, ldap_first_tmplrow, ldap_next_tmplrow, ldap_first_tmplcol, ldap_next_tmplcol – LDAP display template functions |
| SYNOPSIS | <pre> cc[flag ...] file ... -lldap[library ...] #include <lber.h> #include <ldap.h> int ldap_init_templates(char *file, struct ldap_disptmpl **tmplist); int ldap_init_templates_buf(char *buf, unsigned long len, struct ldap_disptmpl **tmplist); void ldap_free_templates(struct ldap_disptmpl *tmplist); struct ldap_disptmpl *ldap_first_disptmpl(struct ldap_disptmpl *tmplist); struct ldap_disptmpl *ldap_next_disptmpl(struct ldap_disptmpl *tmplist, struct ldap_disptmpl *tmpl); struct ldap_disptmpl *ldap_oc2template(char **oclist, struct ldap_disptmpl *tmplist); struct ldap_disptmpl *ldap_name2template(char *name, struct ldap_disptmpl *tmplist); char **ldap_tmplattrs(struct ldap_disptmpl *tmpl, char **includeattrs, int exclude, unsigned long syntaxmask); struct ldap_tmplitem *ldap_first_tmplrow(struct ldap_disptmpl *tmpl); struct ldap_tmplitem *ldap_next_tmplrow(struct ldap_disptmpl *tmpl, struct ldap_tmplitem *row); struct ldap_tmplitem *ldap_first_tmplcol(struct ldap_disptmpl *tmpl, struct ldap_tmplitem *row, struct ldap_tmplitem *col); struct ldap_tmplitem *ldap_next_tmplcol(struct ldap_disptmpl *tmpl, struct ldap_tmplitem *row, struct ldap_tmplitem *col); </pre> |
| DESCRIPTION | <p>These functions provide a standard way to access LDAP entry display templates. Entry display templates provide a standard way for LDAP applications to display directory entries. The general idea is that it is possible</p> |

to map the list of object class values present in an entry to an appropriate display template. Display templates are defined in a configuration file (see `ldaptemplates.conf(4)`). Each display template contains a pre-determined list of items, where each item generally corresponds to an attribute to be displayed. The items contain information and flags that the caller can use to display the attribute and values in a reasonable fashion. Each item has a syntaxid, which are described in the SYNTAX IDS section below. The `ldap_entry2text(3N)` functions use the display template functions and produce text output.

`ldap_init_templates()` reads a sequence of templates from a valid LDAP template configuration file (see `ldaptemplates.conf(4)`). Upon success, 0 is returned, and `tmplist` is set to point to a list of templates. Each member of the list is an `ldap_disptmpl` structure (defined below in the DISPTMPL Structure Elements section).

`ldap_init_templates_buf()` reads a sequence of templates from `buf` (whose size is `buflen`). `buf` should point to the data in the format defined for an LDAP template configuration file (see `ldaptemplates.conf(4)`). Upon success, 0 is returned, and `tmplist` is set to point to a list of templates.

The `LDAP_SET_DISPTMPL_APPDATA()` macro is used to set the value of the `dt_appdata` field in an `ldap_disptmpl` structure. This field is reserved for the calling application to use; it is not used internally.

The `LDAP_GET_DISPTMPL_APPDATA()` macro is used to retrieve the value in the `dt_appdata` field.

The `LDAP_IS_DISPTMPL_OPTION_SET()` macro is used to test a `ldap_disptmpl` structure for the existence of a template option. The options currently defined are: `LDAP_DTmpl_OPT_ADDABLE` (it is appropriate to allow entries of this type to be added), `LDAP_DTmpl_OPT_ALLOWMODRDN` (it is appropriate to offer the "modify rdn" operation), `LDAP_DTmpl_OPT_ALTVIEW` (this template is merely an alternate view of another template, typically used for templates pointed to be an `LDAP_SYN_LINKACTION` item).

`ldap_free_templates()` disposes of the templates allocated by `ldap_init_templates()`.

`ldap_first_disptmpl()` returns the first template in the list `tmplist`. The `tmplist` is typically obtained by calling `ldap_init_templates()`.

`ldap_next_disptmpl()` returns the template after `tmpl` in the template list `tmplist`. A NULL pointer is returned if `tmpl` is the last template in the list.

`ldap_oc2template()` searches `tmplist` for the best template to use to display an entry that has a specific set of `objectClass` values. `oclist` should be a null-terminated array of strings that contains the values of the `objectClass`

attribute of the entry. A pointer to the first template where all of the object classes listed in one of the template's `dt_oclist` elements are contained in `oclist` is returned. A `NULL` pointer is returned if no appropriate template is found.

ldap_tmplattrs() returns a null-terminated array that contains the names of attributes that need to be retrieved if the template *tmpl* is to be used to display an entry. The attribute list should be freed using `ldap_value_free()`. The *includeattrs* parameter contains a null-terminated array of attributes that should always be included (it may be `NULL` if no extra attributes are required). If *syntaxmask* is non-zero, it is used to restrict the attribute set returned. If *exclude* is zero, only attributes where the logical AND of the template item syntax id and the *syntaxmask* is non-zero are included. If *exclude* is non-zero, attributes where the logical AND of the template item syntax id and the *syntaxmask* is non-zero are excluded.

ldap_first_tmplrow() returns a pointer to the first row of items in template *tmpl*.

ldap_next_tmplrow() returns a pointer to the row that follows *row* in template *tmpl*.

ldap_first_tmplcol() returns a pointer to the first item (in the first column) of row *row* within template *tmpl*. A pointer to an `ldap_tmplitem` structure (defined below in the TEMPLITEM Structure Elements section) is returned.

The **LDAP_SET_TMPLITEM_APPDATA()** macro is used to set the value of the `ti_appdata` field in a `ldap_tmplitem` structure. This field is reserved for the calling application to use; it is not used internally.

The **LDAP_GET_TMPLITEM_APPDATA()** macro is used to retrieve the value of the `ti_appdata` field.

The **LDAP_IS_TMPLITEM_OPTION_SET()** macro is used to test a `ldap_tmplitem` structure for the existence of an item option. The options currently defined are: `LDAP_DITEM_OPT_READONLY` (this attribute should not be modified), `LDAP_DITEM_OPT_SORTVALUES` (it makes sense to sort the values), `LDAP_DITEM_OPT_SINGLEVALUED` (this attribute can only hold a single value), `LDAP_DITEM_OPT_VALUEREQUIRED` (this attribute must contain at least one value), `LDAP_DITEM_OPT_HIDEIFEMPTY` (do not show this item if there are no values), and `LDAP_DITEM_OPT_HIDEIFFALSE` (for boolean attributes only: hide this item if the value is `FALSE`).

ldap_next_tmplcol() returns a pointer to the item (column) that follows column `col` within row *row* of template *tmpl*.

The `ldap_disptmpl` structure is defined as:

```

struct ldap_disptmpl {
    \011char          \011*dt_name;
    \011char\011\011\011*dt_pluralname;
    \011char          \011*dt_iconname;
    \011unsigned long \011dt_options;
    \011char          \011*dt_authattrname;
    \011char          \011*dt_defrdnattrname;
    \011char          \011*dt_defaddlocation;
    \011struct ldap_oclist\011*dt_oclist;
    \011struct ldap_adddeflist\011*dt_adddeflist;
    \011struct ldap_tmplitem\011*dt_items;
    \011void\011\011\011*dt_appdata;
    \011struct ldap_disptmpl\011*dt_next;
};

```

The `dt_name` member is the singular name of the template. The `dt_pluralname` is the plural name. The `dt_iconname` member will contain the name of an icon or other graphical element that can be used to depict entries that correspond to this display template. The `dt_options` contains options which may be tested using the `LDAP_IS_TMPLITEM_OPTION_SET()` macro.

The `dt_authattrname` contains the name of the DN-syntax attribute whose value(s) should be used to authenticate to make changes to an entry. If `dt_authattrname` is NULL, then authenticating as the entry itself is appropriate. The `dt_defrdnattrname` is the name of the attribute that is normally used to name entries of this type, for example, "cn" for person entries. The `dt_defaddlocation` is the distinguished name of an entry below which new entries of this type are typically created (its value is site-dependent).

`dt_oclist` is a pointer to a linked list of object class arrays, defined as:

```

struct ldap_oclist {
    \011char\011\011\011**oc_objclasses;
    \011struct ldap_oclist\011*oc_next;
};

```

These are used by the `ldap_oc2template()` function.

`dt_adddeflist` is a pointer to a linked list of rules for defaulting the values of attributes when new entries are created. The `ldap_adddeflist` structure is defined as:

```

struct ldap_adddeflist {
    \011int\011\011\011ad_source;
    \011char\011\011\011*ad_attrname;
    \011char\011\011\011*ad_value;
    \011struct ldap_adddeflist\011*ad_next;
};

```

TMPLITEM Structure Elements

The `ad_attrname` member contains the name of the attribute whose value this rule sets. If `ad_source` is `LDAP_ADSRC_CONSTANTVALUE` then the `ad_value` member contains the (constant) value to use. If `ad_source` is `LDAP_ADSRC_ADDERSDN` then `ad_value` is ignored and the distinguished name of the person who is adding the new entry is used as the default value for `ad_attrname`.

The `ldap_tmplitem` structure is defined as:

```
struct ldap_tmplitem {
    \011unsigned long\011\011ti_syntaxid;
    \011unsigned long\011\011ti_options;
    \011char\011\011\011*ti_attrname;
    \011char\011\011\011*ti_label;
    \011char\011\011\011**ti_args;
    \011struct ldap_tmplitem\011*ti_next_in_row;
    \011struct ldap_tmplitem\011*ti_next_in_col;
    \011void\011\011\011*ti_appdata;
};
```

Syntax IDs

Syntax ids are found in the `ldap_tmplitem` structure element `ti_syntaxid`, and they can be used to determine how to display the values for the attribute associated with an item. The `LDAP_GET_SYN_TYPE()` macro can be used to return a general type from a syntax id. The five general types currently defined are: `LDAP_SYN_TYPE_TEXT` (for attributes that are most appropriately shown as text), `LDAP_SYN_TYPE_IMAGE` (for JPEG or FAX format images), `LDAP_SYN_TYPE_BOOLEAN` (for boolean attributes), `LDAP_SYN_TYPE_BUTTON` (for attributes whose values are to be retrieved and display only upon request, for example, in response to the press of a button, a JPEG image is retrieved, decoded, and displayed), and `LDAP_SYN_TYPE_ACTION` (for special purpose actions such as "search for the entries where this entry is listed in the `seeAlso` attribute").

The `LDAP_GET_SYN_OPTIONS` macro can be used to retrieve an unsigned long bitmap that defines options. The only currently defined option is `LDAP_SYN_OPT_DEFER`, which (if set) implies that the values for the attribute should not be retrieved until requested.

There are sixteen distinct syntax ids currently defined. These generally correspond to one or more X.500 syntaxes.

`LDAP_SYN_CASEIGNORESTR` is used for text attributes which are simple strings whose case is ignored for comparison purposes.

LDAP_SYN_MULTILINESTR is used for text attributes which consist of multiple lines, for example, postalAddress , homePostalAddress , multilineDescription , or any attributes of syntax caseIgnoreList .

LDAP_SYN_RFC822ADDR is used for case ignore string attributes that are RFC-822 conformant mail addresses, for example, mail.

LDAP_SYN_DN is used for attributes with a Distinguished Name syntax, for example, seeAlso .

LDAP_SYN_BOOLEAN is used for attributes with a boolean syntax.

LDAP_SYN_JPEGIMAGE is used for attributes with a jpeg syntax, for example, jpegPhoto.

LDAP_SYN_JPEGBUTTON is used to provide a button (or equivalent interface element) that can be used to retrieve, decode, and display an attribute of jpeg syntax.

LDAP_SYN_FAXIMAGE is used for attributes with a photo syntax, for example, Photo. These are actually Group 3 Fax (T.4) format images.

LDAP_SYN_FAXBUTTON is used to provide a button (or equivalent interface element) that can be used to retrieve, decode, and display an attribute of photo syntax.

LDAP_SYN_AUDIOBUTTON is used to provide a button (or equivalent interface element) that can be used to retrieve and play an attribute of audio syntax. Audio values are in the "mu law" format, also known as "au" format.

LDAP_SYN_TIME is used for attributes with the UTCTime syntax, for example, lastModifiedTime . The value(s) should be displayed in complete date and time fashion.

LDAP_SYN_DATE is used for attributes with the UTCTime syntax, for example, lastModifiedTime . Only the date portion of the value(s) should be displayed.

LDAP_SYN_LABELEDURL is used for labeledURL attributes.

LDAP_SYN_SEARCHACTION is used to define a search that is used to retrieve related information. If ti_attrname is not NULL , it is assumed to be a boolean attribute which will cause no search to be performed if its value is FALSE . The ti_args structure member will have four strings in it: ti_args[0] should be the name of an attribute whose values are used to help construct a search filter or "-dn" is the distinguished name of the entry being displayed should be used, ti_args[1] should be a filter pattern where any occurrences of "%v" are replaced with the value derived from ti_args[0] , ti_args[2] should be the name of an additional attribute

to retrieve when performing the search, and `ti_args[3]` should be a human-consumable name for that attribute. The `ti_args[2]` attribute is typically displayed along with a list of distinguished names when multiple entries are returned by the search.

`LDAP_SYN_LINKACTION` is used to define a link to another template by name. `ti_args[0]` will contain the name of the display template to use. The **`ldap_name2template()`** function can be used to obtain a pointer to the correct `ldap_disptmpl` structure.

`LDAP_SYN_ADDDNACTION` and `LDAP_SYN_VERIFYDNACTION` are reserved as actions but currently undefined.

ERRORS

The init template functions return `LDAP_TMPL_ERR_VERSION` if *buf* points to data that is newer than can be handled, `LDAP_TMPL_ERR_MEM` if there is a memory allocation problem, `LDAP_TMPL_ERR_SYNTAX` if there is a problem with the format of the templates buffer or file. `LDAP_TMPL_ERR_FILE` is returned by `ldap_init_templates` if the file cannot be read. Other functions generally return `NULL` upon error.

ATTRIBUTES

See **`attributes(5)`** for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|---|
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

`ldap(3N)` , **`ldap_entry2text(3N)`** , **`ldaptemplates.conf(4)`**

NAME ldap_entry2text, ldap_entry2text_search, ldap_entry2html, ldap_entry2html_search, ldap_vals2html, ldap_vals2text – LDAP entry display functions

SYNOPSIS

```
cc[
flag
... ]
file
... -lldap[
library
... ]

#include <lber.h>
#include <ldap.h>

int ldap_entry2text(LDAP *ld, char *buf, LDAPMessage *entry, struct ldap_disptmpl
*tmpl, char **def_attrs, char ***defvals, int ( *writeproc )(), void *writeparm, char *eol, int
rdncount, unsigned long opts);

int ldap_entry2text_search(LDAP *ld, char *dn, char *base, LDAPMessage *entry,
struct ldap_disptmpl *tmplist, char **def_attrs, char ***defvals, int ( *writeproc )(), void
*writeparm, char *eol, int rdncount, unsigned long opts);

int ldap_vals2text(LDAP *ld, char *buf, char **vals, char *label, int labelwidth,
unsigned long syntaxid, int ( *writeproc )(), void *writeparm, char *eol, int rdncount);

int ldap_entry2html(LDAP *ld, char *buf, LDAPMessage *entry, struct ldap_disptmpl
*tmpl, char **def_attrs, char ***defvals, int ( *writeproc )(), void *writeparm, char *eol, int
rdncount, unsigned long opts, char *urlprefix, char *base);

int ldap_entry2html_search(LDAP *ld, char *dn, LDAPMessage *entry, struct
ldap_disptmpl *tmplist, char **def_attrs, char ***defvals, int ( *writeproc )(), void *writeparm,
char *eol, int rdncount, unsigned long opts, char *urlprefix);

int ldap_vals2html(LDAP *ld, char *buf, char **vals, char *label, int labelwidth, unsigned
long syntaxid, int ( *writeproc )(), void *writeparm, char *eol, int rdncount, char *urlprefix);

#define LDAP_DISP_OPT_AUTOLABELWIDTH 0x00000001

#define LDAP_DISP_OPT_HTMLBODYONLY    0x00000002

#define LDAP_DTmpl_BUFSIZ  2048
```

DESCRIPTION

These functions use the LDAP display template functions (see `ldap_disptmpl(3N)` and `ldap_templates.conf(4)`) to produce a plain text or an HyperText Markup Language (HTML) display of an entry or a set of values. Typical plain text output produced for an entry might look like:

```
"Barbara J Jensen, Information Technology Division"
Also Known As:
Babs Jensen
Barbara Jensen
Barbara J Jensen
E-Mail Address:
bjensen@terminator.rs.itd.umich.edu
Work Address:
535 W. William
Ann Arbor, MI 48103
Title:
Mythical Manager, Research Systems
...
```

The exact output produced will depend on the display template configuration. HTML output is similar to the plain text output, but more richly formatted.

ldap_entry2text() produces a text representation of *entry* and writes the text by calling the *writeproc* function. All of the attributes values to be displayed must be present in *entry*; no interaction with the LDAP server will be performed within `ldap_entry2text.ld` is the LDAP pointer obtained by a previous call to `ldap_open`. *writeproc* should be declared as:

```
int writeproc( writeparm, p, len )

void *writeparm;
char *p;
int len;
```

where *p* is a pointer to text to be written and *len* is the length of the text. *p* is guaranteed to be zero-terminated. Lines of text are terminated with the string *eol*. *buf* is a pointer to a buffer of size `LDAP_DTEMPL_BUFSIZ` or larger. If *buf* is NULL then a buffer is allocated and freed internally. *tmpl* is a pointer to the display template to be used (usually obtained by calling `ldap_oc2template`). If *tmpl* is NULL, no template is used and a generic display is produced. *defattrs* is a NULL-terminated array of LDAP attribute names which you wish to provide default values for (only used if *entry* contains no values for the attribute). An array of NULL-terminated arrays of default values corresponding to the attributes should be passed in *defvals*. The *rdncount* parameter is used to limit the number of Distinguished Name (DN) components that are actually displayed for DN attributes. If *rdncount* is zero, all components are shown. *opts* is used to specify output options. The only values currently allowed are zero (default output), `LDAP_DISP_OPT_AUTOLABELWIDTH` which causes the width for labels to be determined based on the longest label in *tmpl*, and `LDAP_DISP_OPT_HTMLBODYONLY`. The `LDAP_DISP_OPT_HTMLBODYONLY` option instructs the library not to include `<HTML>`, `<HEAD>`, `<TITLE>`, and `<BODY>` tags. In other words, an HTML fragment is generated, and the caller is responsible for prepending and appending the appropriate HTML tags to construct a correct HTML document.

ldap_entry2text_search() is similar to `ldap_entry2text`, and all of the like-named parameters have the same meaning except as noted below. If *base* is not `NULL`, it is the search base to use when executing search actions. If it is `NULL`, search action template items are ignored. If *entry* is not `NULL`, it should contain the *objectClass* attribute values for the entry to be displayed. If *entry* is `NULL`, *dn* must not be `NULL`, and `ldap_entry2text_search` will retrieve the *objectClass* values itself by calling `ldap_search_s`.

`ldap_entry2text_search` will determine the appropriate display template to use by calling `ldap_oc2template`, and will call `ldap_search_s` to retrieve any attribute values to be displayed. The *tmplist* parameter is a pointer to the entire list of templates available (usually obtained by calling `ldap_init_templates` or `ldap_init_templates_buf`). If *tmplist* is `NULL`, `ldap_entry2text_search` will attempt to read a load templates from the default template configuration file `ETCDIR/ldaptemplates.conf`.

`ldap_vals2text` produces a text representation of a single set of LDAP attribute values. The *ld*, *buf*, *writeproc*, *writeparm*, *eol*, and *rdncount* parameters are the same as the like-named parameters for `ldap_entry2text`. *vals* is a `NULL`-terminated list of values, usually obtained by a call to `ldap_get_values`. *label* is a string shown next to the values (usually a friendly form of an LDAP attribute name). *labelwidth* specifies the label margin, which is the number of blank spaces displayed to the left of the values. If zero is passed, a default label width is used. *syntaxid* is a display template attribute syntax identifier (see `ldap_disptmpl(3N)` for a list of the pre-defined `LDAP_SYN_...` values).

`ldap_entry2html` produces an HTML representation of *entry*. It behaves exactly like `ldap_entry2text(3N)`, except for the formatted output and the addition of two parameters. *urlprefix* is the starting text to use when constructing an LDAP URL. The default is the string `ldap:///`. The second additional parameter, *base*, the search base to use when executing search actions. If it is `NULL`, search action template items are ignored.

`ldap_entry2html_search` behaves exactly like `ldap_entry2text_search(3N)`, except HTML output is produced and one additional parameter is required. *urlprefix* is the starting text to use when constructing an LDAP URL. The default is the string `ldap:///`.

`ldap_vals2html` behaves exactly like `ldap_vals2text`, except HTML output is and one additional parameter is required. *urlprefix* is the starting text to use when constructing an LDAP URL. The default is the string `ldap:///`.

ERRORS

These functions all return an LDAP error code (`LDAP_SUCCESS` is returned if no error occurs). See `ldap_error(3N)` for details. The *ld_errno* field of the *ld* parameter is also set to indicate the error.

FILES

ETCDIR/ldaptemplates.conf

ATTRIBUTESSee **attributes(5)** for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|--|
| Availability | SUNWlldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO**ldap(3N)** , **ldap_disptmpl(3N)** , **ldaptemplates.conf(4)**

| | |
|--------------------|---|
| NAME | ldap_error, ldap_perror, ldap_result2error, ldap_errlist, ldap_err2string – LDAP protocol error handling functions |
| SYNOPSIS | <pre> cc[flag ...] file ... -lldap[library ...] #include <lber.h> #include <ldap.h> struct ldap_error(int e_code, char *e_reason); struct ldaperror ldap_errlist[]; char * ldap_err2string(int err); void ldap_perror(LDAP *ld, char *s); int ldap_result2error(LDAP *ld, LDAPMessage *res, int freeit); </pre> |
| DESCRIPTION | <p>These functions provide interpretation of the various error codes returned by the LDAP protocol and LDAP library functions and assigned to the <i>ld_errno</i> field in the <i>ld</i> structure.</p> <p>The ldap_result2error() function takes <i>res</i>, a result as produced by ldap_result(3N) or ldap_search_s(3N), and returns the corresponding error code. Possible error codes are listed below. If the <i>freet</i> parameter is non zero it indicates that the <i>res</i> parameter should be freed by a call to ldap_msgfree(3N) after the error code has been extracted. The <i>ld_errno</i> field in <i>ld</i> is set and returned.</p> <p>The returned value can be passed to ldap_err2string() or looked up in <i>ldap_errlist[]</i> to get a text description of the message. The string returned from ldap_err2string() is a pointer to a static area that should not be modified. The last element in the <i>ldap_errlist[]</i> array is signaled by an error code of -1.</p> <p>The ldap_perror() function can be called to print an indication of the error on standard error, similar to the way perror(3C) works.</p> |
| ERRORS | <p>The possible values for an ldap error code are:</p> <pre> LDAP_SUCCESS The request was successful. </pre> |

| | |
|--------------------------------|--|
| LDAP_OPERATIONS_ERROR | An operations error occurred. |
| LDAP_PROTOCOL_ERROR | A protocol violation was detected. |
| LDAP_TIMELIMIT_EXCEEDED | An LDAP time limit was exceeded. |
| LDAP_SIZELIMIT_EXCEEDED | An LDAP size limit was exceeded. |
| LDAP_COMPARE_FALSE | A compare operation returned false. |
| LDAP_COMPARE_TRUE | A compare operation returned true. |
| LDAP_STRONG_AUTH_NOT_SUPPORTED | The LDAP server does not support strong authentication. |
| LDAP_STRONG_AUTH_REQUIRED | Strong authentication is required for the operation. |
| LDAP_PARTIAL_RESULTS | Partial results only returned. |
| LDAP_NO_SUCH_ATTRIBUTE | The attribute type specified does not exist in the entry. |
| LDAP_UNDEFINED_TYPE | The attribute type specified is invalid. |
| LDAP_INAPPROPRIATE_MATCHING | Filter type not supported for the specified attribute. |
| LDAP_CONSTRAINT_VIOLATION | An attribute value specified violates some constraint (for example, a postalAddress has too many lines, or a line that is too long). |
| LDAP_TYPE_OR_VALUE_EXISTS | An attribute type or attribute value specified already exists in the entry. |
| LDAP_INVALID_SYNTAX | An invalid attribute value was specified. |
| LDAP_NO_SUCH_OBJECT | The specified object does not exist in The Directory. |
| LDAP_ALIAS_PROBLEM | An alias in The Directory points to a nonexistent entry. |

| | |
|-----------------------------|--|
| LDAP_INVALID_DN_SYNTAX | A syntactically invalid DN was specified. |
| LDAP_IS_LEAF | The object specified is a leaf. |
| LDAP_ALIAS_DEREF_PROBLEM | A problem was encountered when dereferencing an alias. |
| LDAP_INAPPROPRIATE_AUTH | Inappropriate authentication was specified (for example, LDAP_AUTH_SIMPLE was specified and the entry does not have a userPassword attribute). |
| LDAP_INVALID_CREDENTIALS | Invalid credentials were presented (for example, the wrong password). |
| LDAP_INSUFFICIENT_ACCESS | The user has insufficient access to perform the operation. |
| LDAP_BUSY | The DSA is busy. |
| LDAP_UNAVAILABLE | The DSA is unavailable. |
| LDAP_UNWILLING_TO_PERFORM | The DSA is unwilling to perform the operation. |
| LDAP_LOOP_DETECT | A loop was detected. |
| LDAP_NAMING_VIOLATION | A naming violation occurred. |
| LDAP_OBJECT_CLASS_VIOLATION | An object class violation occurred (for example, a "must" attribute was missing from the entry). |
| LDAP_NOT_ALLOWED_ON_NONLEAF | The operation is not allowed on a nonleaf object. |
| LDAP_NOT_ALLOWED_ON_RDN | The operation is not allowed on an RDN. |
| LDAP_ALREADY_EXISTS | The entry already exists. |
| LDAP_NO_OBJECT_CLASS_MODS | Object class modifications are not allowed. |

| | |
|---------------------|--|
| LDAP_OTHER | An unknown error occurred. |
| LDAP_SERVER_DOWN | The LDAP library can't contact the LDAP server. |
| LDAP_LOCAL_ERROR | Some local error occurred. This is usually a failed malloc. |
| LDAP_ENCODING_ERROR | An error was encountered encoding parameters to send to the LDAP server. |
| LDAP_DECODING_ERROR | An error was encountered decoding a result from the LDAP server. |
| LDAP_TIMEOUT | A timelimit was exceeded while waiting for a result. |
| LDAP_AUTH_UNKNOWN | The authentication method specified to ldap_bind() is not known. |
| LDAP_FILTER_ERROR | An invalid filter was supplied to ldap_search() (for example, unbalanced parentheses). |
| LDAP_PARAM_ERROR | An ldap function was called with a bad parameter (for example, a NULL ld pointer, etc.). |
| LDAP_NO_MEMORY | An memory allocation (for example, malloc(3N)) call failed in an ldap library function. |

ATTRIBUTES

See **attributes(5)** for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|--|
| Availability | SUNWlldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

ldap(3N) , **perror(3C)**

| | |
|--------------------|--|
| NAME | ldap_first_attribute, ldap_next_attribute – step through LDAP entry attributes |
| SYNOPSIS | <pre>cc[flag ...] file ... -lldap[library ...] #include <lber.h> #include <ldap.h> char * ldap_first_attribute(LDAP *ld, LDAPMessage *entry, BerElement **berptr); char * ldap_next_attribute(LDAP *ld, LDAPMessage *entry, BerElement *ber);</pre> |
| DESCRIPTION | <p>The ldap_first_attribute() and ldap_next_attribute() functions are used to step through the attributes in an LDAP entry. ldap_first_attribute() takes an <i>entry</i> as returned by ldap_first_entry(3N) or ldap_next_entry(3N) and returns a pointer to a per-connection buffer containing the first attribute type in the entry. The return value should be treated as if it is a pointer to a static area (that is, strdup(3C) if you want to save it).</p> <p>It also returns, in <i>berptr</i>, a pointer to a <code>BerElement</code> it has allocated to keep track of its current position. This pointer should be passed to subsequent calls to ldap_next_attribute() and is used to effectively step through the entry's attributes. This pointer is freed by ldap_next_attribute() when there are no more attributes (that is, when ldap_next_attribute() returns <code>NULL</code>). Otherwise, the caller is responsible for freeing the <code>BerElement</code> pointed to by <i>berptr</i> when it is no longer needed by calling ber_free(3N). When calling ber_free(3N) in this instance, be sure the second argument is '0'.</p> <p>The attribute names returned are suitable for inclusion in a call to ldap_get_values(3N) to retrieve the attribute's values.</p> |
| ERRORS | If an error occurs, <code>NULL</code> is returned and the <code>ld_errno</code> field in the <i>ld</i> parameter is set to indicate the error. See ldap_error(3N) for a description of possible error codes. |
| ATTRIBUTES | See attributes(5) for a description of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|---|
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO `ldap(3N)` , `ldap_first_entry(3N)` , `ldap_get_values(3N)` ,
`ldap_error(3N)`

NOTES The `ldap_first_attribute()` function mallocs memory that may need to be freed by the caller via `ber_free(3N)` .

| | |
|--------------------|---|
| NAME | ldap_first_entry, ldap_next_entry, ldap_count_entries, ldap_count_references, ldap_first_reference, ldap_next_reference - LDAP entry parsing and counting functions |
| SYNOPSIS | <pre> cc[flag ...] file ... -lldap[library ...] #include <lber.h> #include <ldap.h> LDAPMessage * ldap_first_entry(LDAP *ld, LDAPMessage *result); LDAPMessage * ldap_next_entry(LDAP *ld, LDAPMessage *entry); ldap_count_entries(LDAP *ld, LDAPMessage *result); LDAPMessage * ldap_first_reference(LDAP *ld, LDAPMessage *res); LDAPMessage * ldap_next_reference(LDAP *ld, LDAPMessage *res); int ldap_count_references(LDAP *ld, LDAPMessage *res); </pre> |
| DESCRIPTION | <p>These functions are used to parse results received from <code>ldap_result(3N)</code> or the synchronous LDAP search operation functions <code>ldap_search_s(3N)</code> and <code>ldap_search_st(3N)</code>.</p> <p>The <code>ldap_first_entry()</code> function is used to retrieve the first entry in a chain of search results. It takes the <i>result</i> as returned by a call to <code>ldap_result(3N)</code> or <code>ldap_search_s(3N)</code> or <code>ldap_search_st(3N)</code> and returns a pointer to the first entry in the result.</p> <p>This pointer should be supplied on a subsequent call to <code>ldap_next_entry()</code> to get the next entry, the result of which should be supplied to the next call to <code>ldap_next_entry()</code>, etc. <code>ldap_next_entry()</code> will return <code>NULL</code> when there are no more entries. The entries returned from these calls are used in calls to the functions described in <code>ldap_get_dn(3N)</code>, <code>ldap_first_attribute(3N)</code>, <code>ldap_get_values(3N)</code>, etc.</p> <p>A count of the number of entries in the search result can be obtained by calling <code>ldap_count_entries()</code>.</p> <p><code>ldap_first_reference()</code> and <code>ldap_next_reference()</code> are used to step through and retrieve the list of continuation references from a search result chain.</p> |

The **ldap_count_references()** function is used to count the number of references that are contained in and remain in a search result chain.

ERRORS

If an error occurs in **ldap_first_entry()** or **ldap_next_entry()**, `NULL` is returned and the `ld_errno` field in the `ld` parameter is set to indicate the error. If an error occurs in **ldap_count_entries()**, `-1` is returned, and `ld_errno` is set appropriately. See **ldap_error(3N)** for a description of possible error codes.

ATTRIBUTES

See **attributes(5)** for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|---|
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

ldap(3N), **ldap_result(3N)**, **ldap_search(3N)**,
ldap_first_attribute(3N), **ldap_get_values(3N)**,
ldap_get_dn(3N)

| | |
|----------------------|--|
| NAME | ldap_first_message, ldap_count_message, ldap_next_message, ldap_msgtype – LDAP message processing functions |
| SYNOPSIS | <pre> cc[flag ...] file ... -lldap[library ...] #include <lber.h> #include <ldap.h> int ldap_count_messages(LDAP *ld, LDAPMessage *res); LDAPMessage * ldap_first_message(LDAP *ld, LDAPMessage *res); LDAPMessage * ldap_next_message(LDAP *ld, LDAPMessage *msg); int ldap_msgtype(LDAPMessage *res); </pre> |
| DESCRIPTION | <p>ldap_count_messages() is used to count the number of messages that remain in a chain of results if called with a message, entry, or reference returned by ldap_first_message() , ldap_next_message() , ldap_first_entry() , ldap_next_entry() , ldap_first_reference() , and ldap_next_reference()</p> <p>ldap_first_message() and ldap_next_message() functions are used to step through the list of messages in a result chain returned by ldap_result() .</p> <p>ldap_msgtype() function returns the type of an LDAP message.</p> |
| RETURN VALUES | <p>ldap_first_message() and ldap_next_message() return <code>LDAPMessage</code> which can include referral messages, entry messages and result messages.</p> <p>ldap_count_messages() returns the number of messages contained in a chain of results.</p> |
| ERRORS | <p>ldap_first_message() and ldap_next_message() return <code>NULL</code> when no more messages exist. <code>NULL</code> is also returned if an error occurs while stepping through the entries, in which case the error parameters in the session handle <code>ld</code> will be set to indicate the error.</p> |
| ATTRIBUTES | See attributes(5) for a description of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|---|
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO `ldap_error(3N)` , `ldap_result(3N)` , `attributes(5)`

| | |
|--------------------|---|
| NAME | ldap_friendly, ldap_friendly_name, ldap_free_friendlymap – LDAP attribute remapping functions |
| SYNOPSIS | <pre>cc[flag ...] file ... -lldap[library ...] #include <lber.h> #include <ldap.h> char * ldap_friendly_name(char *filename, char *name, FriendlyMap **map); void ldap_free_friendlymap(FriendlyMap **map);</pre> |
| DESCRIPTION | <p>This function is used to map one set of strings to another. Typically, this is done for country names, to map from the two-letter country codes to longer more readable names. The mechanism is general enough to be used with other things, though.</p> <p><i>filename</i> is the name of a file containing the unfriendly to friendly mapping, <i>name</i> is the unfriendly name to map to a friendly name, and <i>map</i> is a result-parameter that should be set to NULL on the first call. It is then used to hold the mapping in core so that the file need not be read on subsequent calls.</p> <p>For example:</p> <pre> FriendlyMap *map = NULL; printf("unfriendly %s => friendly %s\ ", name, ldap_friendly_name("ETCDIR/ldapfriendly", name, &map));</pre> <p>The mapping file should contain lines like this: unfriendlyname\\tfriendlyname. Lines that begin with a '#' character are comments and are ignored.</p> <p>The ldap_free_friendlymap() call is used to free structures allocated by ldap_friendly_name() when no more calls to ldap_friendly_name() are to be made.</p> |
| ERRORS | NULL is returned by ldap_friendly_name() if there is an error opening <i>filename</i> , or if the file has a bad format, or if the <i>map</i> parameter is NULL. |
| FILES | ETCDIR/ldapfriendly.conf |

ATTRIBUTES

See **attributes(5)** for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|--|
| Availability | SUNWlldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

ldap(3N)

| | |
|--------------------|---|
| NAME | ldap_get_dn, ldap_explode_dn, ldap_dn2ufn, ldap_is_dns_dn, ldap_explode_dns, ldap_dns_to_dn - LDAP DN handling functions |
| SYNOPSIS | <pre>cc[flag ...] file ... -lldap[library ...] #include <lber.h> #include <ldap.h> char * ldap_get_dn(LDAP *ld, LDAPMessage *entry); char ** ldap_explode_dn(char *dn, int notypes); char * ldap_dn2ufn(char *dn); int ldap_is_dns_dn(char *dn); char ** ldap_explode_dns(char *dn); char * ldap_dns_to_dn(char *dns_name, int *nameparts);</pre> |
| DESCRIPTION | <p>These functions allow LDAP entry names (Distinguished Names, or DN's) to be obtained, parsed, converted to a user-friendly form, and tested. A DN has the form described in RFC 1779 <i>A String Representation of Distinguished Names</i>, unless it is an experimental DNS-style DN which takes the form of an RFC 822 mail address.</p> <p>The ldap_get_dn() function takes an <i>entry</i> as returned by ldap_first_entry(3N) or ldap_next_entry(3N) and returns a copy of the entry's DN. Space for the DN will have been obtained via malloc(3C), and should be freed by the caller by a call to free(3C).</p> <p>The ldap_explode_dn() function takes a DN as returned by ldap_get_dn() and breaks it up into its component parts. Each part is known as a Relative Distinguished Name, or RDN. ldap_explode_dn() returns a NULL-terminated array, each component of which contains an RDN from the DN. The <i>notypes</i> parameter is used to request that only the RDN values be returned, not their types. For example, the DN "cn=Bob, c=US" would return as either { "cn=Bob", "c=US", NULL } or { "Bob", "US", NULL }, depending on whether <i>notypes</i> was 0 or 1, respectively. The result can be freed by calling ldap_value_free(3N).</p> <p>ldap_dn2ufn() is used to turn a DN as returned by ldap_get_dn() into a more user-friendly form, stripping off type names. See RFC 1781 "Using the Directory to Achieve User Friendly Naming" for more details on the UFN</p> |

format. The space for the UFN returned is obtained by a call to `malloc(3C)`, and the user is responsible for freeing it via a call to `free(3C)`.

ldap_is_dns_dn() returns non-zero if the dn string is an experimental DNS-style DN (generally in the form of an RFC 822 e-mail address). It returns zero if the dn appears to be an RFC 1779 format DN.

ldap_explode_dns() takes a DNS-style DN and breaks it up into its component parts. **ldap_explode_dns()** returns a NULL-terminated array. For example, the DN "mcs.umich.edu" will return { "mcs", "umich", "edu", NULL }. The result can be freed by calling `ldap_value_free(3N)`.

ldap_dns_to_dn() converts a DNS domain name into an X.500 distinguished name. A string distinguished name and the number of nameparts is returned.

ERRORS

If an error occurs in **ldap_get_dn()**, NULL is returned and the `ld_errno` field in the `ld` parameter is set to indicate the error. See `ldap_error(3N)` for a description of possible error codes. **ldap_explode_dn()**, **ldap_explode_dns()** and **ldap_dn2ufn()** will return NULL with `errno(3C)` set appropriately in case of trouble.

If an error in **ldap_dns_to_dn()** is encountered zero is returned. The caller should free the returned string if it is non-zero.

ATTRIBUTES

See `attributes(5)` for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|---|
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

`ldap(3N)`, `ldap_first_entry(3N)`, `ldap_error(3N)`, `ldap_value_free(3N)`

NOTES

These functions allocate memory that the caller must free.

| | |
|--------------------|--|
| NAME | ldap_getfilter, ldap_init_getfilter, ldap_init_getfilter_buf, ldap_getfilter_free, ldap_getfirstfilter, ldap_getnextfilter, ldap_build_filter – LDAP filter generating functions |
| SYNOPSIS | <pre> cc[flag ...] file ... -lldap[library ...] #include <lber.h> #include <ldap.h> #define LDAP_FILT_MAXSIZE 0111024 LDAPFiltDesc * ldap_init_getfilter(char *file); LDAPFiltDesc * ldap_init_getfilter_buf(char *buf, long buflen); ldap_getfilter_free(LDAPFiltDesc *lfdp); LDAPFiltInfo * ldap_getfirstfilter(LDAPFiltDesc *lfdp, char *tagpat, char *value); LDAPFiltInfo * ldap_getnextfilter(LDAPFiltDesc *lfdp); void ldap_setfilteraffixes(LDAPFiltDesc *lfdp, char *prefix, char *suffix); void ldap_build_filter(char *buf, unsigned long buflen, char *pattern, char *prefix, char *suffix, char *attr, char *value, char **valwords); </pre> |
| DESCRIPTION | <p>These functions are used to generate filters to be used in <code>ldap_search(3N)</code> or <code>ldap_search_s(3N)</code>. Either <code>ldap_init_getfilter</code> or <code>ldap_init_getfilter_buf</code> must be called prior to calling any of the other functions except <code>ldap_build_filter</code>.</p> <p><code>ldap_init_getfilter()</code> takes a file name as its only argument. The contents of the file must be a valid LDAP filter configuration file (see <code>ldapfilter.conf(4)</code>). If the file is successfully read, a pointer to an <code>LDAPFiltDesc</code> is returned. This is an opaque object that is passed in subsequent get filter calls.</p> <p><code>ldap_init_getfilter_buf()</code> reads from <code>buf</code> (whose length is <code>buflen</code>) the LDAP filter configuration information. <code>buf</code> must point to the contents of a valid LDAP filter configuration file (see <code>ldapfilter.conf(4)</code>). If the filter configuration information is successfully read, a pointer to an <code>LDAPFiltDesc</code> is returned. This is an opaque object that is passed in subsequent get filter calls.</p> |

ldap_getfilter_free() deallocates the memory consumed by `ldap_init_getfilter`. Once it is called, the `LDAPFiltDesc` is no longer valid and cannot be used again.

ldap_getfirstfilter() retrieves the first filter that is appropriate for *value*. Only filter sets that have tags that match the regular expression *tagpat* are considered. `ldap_getfirstfilter` returns a pointer to an `LDAPFiltInfo` structure, which contains a filter with *value* inserted as appropriate in `lfi_filter`, a text match description in `lfi_desc`, `lfi_scope` set to indicate the search scope, and `lfi_isexact` set to indicate the type of filter. NULL is returned if no matching filters are found. `lfi_scope` will be one of `LDAP_SCOPE_BASE`, `LDAP_SCOPE_ONELEVEL`, or `LDAP_SCOPE_SUBTREE`. `lfi_isexact` will be zero if the filter has any '~' or '*' characters in it and non-zero otherwise.

ldap_getnextfilter() retrieves the next appropriate filter in the filter set that was determined when `ldap_getfirstfilter` was called. It returns NULL when the list has been exhausted.

ldap_setfilteraffixes() sets a *prefix* to be prepended and a *suffix* to be appended to all filters returned in the future.

ldap_build_filter() constructs an LDAP search filter in *buf*. *buflen* is the size, in bytes, of the largest filter *buf* can hold. A pattern for the desired filter is passed in *pattern*. Where the string *%a* appears in the pattern it is replaced with *attr*. *prefix* is pre-pended to the resulting filter, and *suffix* is appended. Either can be NULL (in which case they are not used). *value* and *valwords* are used when the string *%v* appears in *pattern*. See `ldapfilter.conf(4)` for a description of how *%v* is handled.

ERRORS

NULL is returned by `ldap_init_getfilter` if there is an error reading *file*. NULL is returned by `ldap_getfirstfilter` and `ldap_getnextfilter` when there are no more appropriate filters to return.

FILES

`ETCDIR/ldapfilter.conf` LDAP filtering routine configuration file.

ATTRIBUTES

See `attributes(5)` for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|---|
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

`ldap(3N)`, `ldapfilter.conf(4)`

NOTES

The return values for all of these functions are declared in the `<ldap.h>` header file. Some functions may allocate memory which must be freed by the calling application.

| | |
|--------------------|---|
| NAME | ldap_get_values, ldap_get_values_len, ldap_count_values, ldap_count_values_len, ldap_value_free, ldap_value_free_len – LDAP attribute value handling functions |
| SYNOPSIS | <pre> cc[flag ...] file ... -lldap[library ...] #include <lber.h> #include <ldap.h> char ** ldap_get_values(LDAP *ld, LDAPMessage *entry, char *attr); struct berval ** ldap_get_values_len(LDAP *ld, LDAPMessage *entry, char *attr); ldap_count_values(char **vals); ldap_count_values_len(struct berval **vals); ldap_value_free(char **vals); ldap_value_free_len(struct berval **vals); </pre> |
| DESCRIPTION | <p>These functions are used to retrieve and manipulate attribute values from an LDAP entry as returned by <code>ldap_first_entry(3N)</code> or <code>ldap_next_entry(3N)</code>. <code>ldap_get_values()</code> takes the <code>entry</code> and the attribute <code>attr</code> whose values are desired and returns a NULL-terminated array of the attribute's values. <code>attr</code> may be an attribute type as returned from <code>ldap_first_attribute(3N)</code> or <code>ldap_next_attribute(3N)</code>, or if the attribute type is known it can simply be given.</p> <p>The number of values in the array can be counted by calling <code>ldap_count_values()</code>. The array of values returned can be freed by calling <code>ldap_value_free()</code>.</p> <p>If the attribute values are binary in nature, and thus not suitable to be returned as an array of char *'s, the <code>ldap_get_values_len()</code> function can be used instead. It takes the same parameters as <code>ldap_get_values()</code>, but returns a NULL-terminated array of pointers to berval structures, each containing the length of and a pointer to a value.</p> <p>The number of values in the array can be counted by calling <code>ldap_count_values_len()</code>. The array of values returned can be freed by calling <code>ldap_value_free_len()</code>.</p> |

ERRORS If an error occurs in `ldap_get_values()` or `ldap_get_values_len()`, `NULL` returned and the `ld_errno` field in the `ld` parameter is set to indicate the error. See `ldap_error(3N)` for a description of possible error codes.

ATTRIBUTES See `attributes(5)` for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|---|
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO `ldap(3N)`, `ldap_first_entry(3N)`, `ldap_first_attribute(3N)`, `ldap_error(3N)`

NOTES These functions allocates memory that the caller must free.

| | |
|--------------------|---|
| NAME | ldap_modify, ldap_modify_s, ldap_mods_free, ldap_modify_ext, ldap_modify_ext_s - LDAP entry modification functions |
| SYNOPSIS | <pre> cc[flag ...] file ... -lldap[library ...] #include <lber.h> #include <ldap.h> int ldap_modify(LDAP *ld, char *dn, LDAPMod *mods []); int ldap_modify_s(LDAP *ld, char *dn, LDAPMod *mods []); void ldap_mods_free(LDAPMod **mods, int freemods); int ldap_modify_ext(LDAP *ld, char *dn, LDAPMod **mods, LDAPControl **serverctrls, LDAPControl **clientctrls, int *msgidp); int ldap_modify_ext_s(LDAP *ld, char *dn, LDAPMod **mods, LDAPControl **serverctrls, LDAPControl **clientctrls); </pre> |
| DESCRIPTION | <p>The function ldap_modify_s() is used to perform an LDAP modify operation. <i>dn</i> is the DN of the entry to modify, and <i>mods</i> is a null-terminated array of modifications to make to the entry. Each element of the <i>mods</i> array is a pointer to an LDAPMod structure, which is defined below.</p> <pre> \011typedef struct ldapmod { \011 int mod_op; \011 char *mod_type; \011 union { \011\011char **modv_strvals; \011\011struct berval **modv_bvals; \011 } mod_vals; \011 struct ldapmod *mod_next; \011} LDAPMod; \011#define mod_values mod_vals.modv_strvals \011#define mod_bvalues mod_vals.modv_bvals </pre> <p>The <i>mod_op</i> field is used to specify the type of modification to perform and should be one of LDAP_MOD_ADD , LDAP_MOD_DELETE , or LDAP_MOD_REPLACE . The <i>mod_type</i> and <i>mod_values</i> fields specify the attribute type to modify and a null-terminated array of values to add, delete, or replace respectively. The <i>mod_next</i> field is used only by the LDAP server and may be ignored by the client.</p> |

If you need to specify a non-string value (for example, to add a photo or audio attribute value), you should set *mod_op* to the logical OR of the operation as above (for example, LDAP_MOD_REPLACE) and the constant LDAP_MOD_BVALUES. In this case, *mod_bvalues* should be used instead of *mod_values*, and it should point to a null-terminated array of struct berval, as defined in <lber.h>.

For LDAP_MOD_ADD modifications, the given values are added to the entry, creating the attribute if necessary. For LDAP_MOD_DELETE modifications, the given values are deleted from the entry, removing the attribute if no values remain. If the entire attribute is to be deleted, the *mod_values* field should be set to NULL. For LDAP_MOD_REPLACE modifications, the attribute will have the listed values after the modification, having been created if necessary. All modifications are performed in the order in which they are listed.

ldap_modify_s() returns the LDAP error code resulting from the modify operation. This code can be interpreted by **ldap_perror(3N)**.

The **ldap_modify()** operation works the same way as **ldap_modify_s()**, except that it is asynchronous, returning the message id of the request it initiates, or -1 on error. The result of the operation can be obtained by calling **ldap_result(3N)**.

ldap_mods_free() can be used to free each element of a NULL-terminated array of mod structures. If *freemods* is non-zero, the *mods* pointer itself is freed as well.

The **ldap_modify_ext()** function initiates an asynchronous modify operation and returns LDAP_SUCCESS if the request was successfully sent to the server, or else it returns a LDAP error code if not (see **ldap_error(3n)**). If successful, **ldap_modify_ext()** places the message id of the request in **msgidp*. A subsequent call to **ldap_result(3N)**, can be used to obtain the result of the add request.

The **ldap_modify_ext_s()** function initiates a synchronous modify operation and returns the result of the operation itself.

ERRORS

ldap_modify_s() returns an ldap error code, either LDAP_SUCCESS or an error (see **ldap_error(3N)**).

ldap_modify() returns -1 in case of trouble, setting the *ld_errno* field of *ld*.

ATTRIBUTES

See **attributes(5)** for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|---|
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO `ldap(3N)` , `ldap_error(3N)` , `ldap_add(3N)`

| | |
|--------------------|---|
| NAME | ldap_modrdn, ldap_modrdn_s, ldap_modrdn2, ldap_modrdn2_s, ldap_rename, ldap_rename_s - modify LDAP entry RDN |
| SYNOPSIS | <pre> cc[flag ...] file ... -lldap[library ...] #include <lber.h> #include <ldap.h> int ldap_modrdn(LDAP **ld, char **dn, char **newrdn); int ldap_modrdn_s(LDAP **ld, char **dn, char **newrdn, int deleteoldrdn); int ldap_modrdn2(LDAP **ld, char **dn, char **newrdn, int deleteoldrdn); int ldap_modrdn2_s(LDAP **ld, char **dn, char **newrdn, int deleteoldrdn); int ldap_rename(LDAP *ld, char *dn, char *newrdn, char *newparent, int deleteoldrdn, LDAPControl **serverctrls, LDAPControl **clientctrls, int *msgidp); int ldap_rename_s(LDAP *ld, char *dn, char *newrdn, char *newparent, int deleteoldrdn, LDAPControl **serverctrls, LDAPControl **clientctrls); </pre> |
| DESCRIPTION | <p>The ldap_modrdn() and ldap_modrdn_s() functions perform an LDAP modify RDN (Relative Distinguished Name) operation. They both take <i>dn</i>, the DN of the entry whose RDN is to be changed, and <i>newrdn</i>, the new RDN to give the entry. The old RDN of the entry is never kept as an attribute of the entry. ldap_modrdn() is asynchronous, returning the message id of the operation it initiates. ldap_modrdn_s() is synchronous, returning the LDAP error code indicating the success or failure of the operation. Use of these functions is deprecated. Use the versions described below instead.</p> <p>The ldap_modrdn2() and ldap_modrdn2_s() functions also perform an LDAP modify RDN operation, taking the same parameters as above. In addition, they both take the <i>deleteoldrdn</i> parameter which is used as a boolean value to indicate whether the old RDN values should be deleted from the entry or not.</p> <p>The ldap_modrdn_s() routine is deprecated and the ldap_rename() and ldap_rename_s() routines are used instead.</p> <p>The ldap_rename(), ldap_rename_s() routines are used to change the name, that is, the rdn of an entry. These routines deprecate ldap_modrdn() and ldap_modrdn_s().</p> |

The **ldap_rename()** and **ldap_rename_s()** functions both support LDAPv3 server controls and client controls.

ERRORS

The synchronous (**_s**) versions of these functions return an LDAP error code, either LDAP_SUCCESS or an error (see **ldap_error(3N)**).

The asynchronous versions return - 1 in case of trouble, setting the **ld_errno** field of **ld** . See **ldap_error(3N)** for more details. Use **ldap_result(3N)** to determine a particular unsuccessful result.

ATTRIBUTES

See **attributes(5)** for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|--|
| Availability | SUNWlldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

ldap(3N) , **ldap_error(3N)**

| | |
|--------------------|--|
| NAME | ldap_open, ldap_init – initialize the LDAP library and open a connection to an LDAP server |
| SYNOPSIS | <pre>cc[flag ...] file ... -lldap[library ...] #include <lber.h> #include <ldap.h> LDAP * ldap_open(char *host, int port); LDAP * ldap_init(char *host, int port);</pre> |
| DESCRIPTION | <p>ldap_open() opens a connection to an LDAP server and allocates an LDAP structure which is used to identify the connection and to maintain per-connection information. ldap_init() allocates an LDAP structure but does not open an initial connection. One of these two functions must be called before any operations are attempted.</p> <p>ldap_open() takes <i>host</i>, the hostname on which the LDAP server is running, and <i>port</i>, the port number to which to connect. If the default IANA-assigned port of 389 is desired, LDAP_PORT should be specified for <i>port</i>. The <i>host</i> parameter may contain a blank-separated list of hosts to try to connect to, and each host may optionally be of the form <i>host:port</i>. If present, the <i>:port</i> overrides the <i>port</i> parameter to ldap_open(). Upon successfully making a connection to an LDAP server, ldap_open() returns a pointer to an LDAP structure (defined below), which should be passed to subsequent calls to ldap_bind(), ldap_search(), etc. Certain fields in the LDAP structure can be set to indicate size limit, time limit, and how aliases are handled during operations. See <ldap.h> for more details.</p> <pre>\011typedef struct ldap { \011\011/* ... other stuff you should not mess with ... */ \011\011char\011\011ld_lberoptions; \011\011int\011\011ld_deref; \011#define LDAP_DEREF_NEVER\0110 \011#define LDAP_DEREF_SEARCHING\0111 \011#define LDAP_DEREF_FINDING\0112 \011#define LDAP_DEREF_ALWAYS\0113 \011\011int\011\011ld_timelimit; \011\011int\011\011ld_sizelimit; \011#define LDAP_NO_LIMIT\011\0110 \011\011int\011\011ld_errno; \011\011char\011\011*ld_error; \011\011char\011\011*ld_matched;</pre> |

```
\011\011int\011\011ld_refhoplmit;
\011\011unsigned long\011ld_options;
\011#define LDAP_OPT_REFERRALS      0x00000002\011/* set by default */
\011#define LDAP_OPT_RESTART\0110x00000004
\011\011/* ... other stuff you should not mess with ... */
\011} LDAP;
```

`ldap_init()` acts just like `ldap_open()`, but does not open a connection to the LDAP server. The actual connection open will occur when the first operation is attempted. At this time, `ldap_init()` should only be used if the LDAP library is compiled with `LDAP_REFERRALS` defined.

OPTIONS

Options that affect a particular LDAP instance may be set by modifying the `ld_options` field in the LDAP structure. This field is set to `LDAP_OPT_REFERRALS` in `ldap_open()` and `ldap_init()`, which causes the library to automatically follow referrals to other servers that may be returned in response to an LDAP operation.

The other supported option is `LDAP_OPT_RESTART`, which if set will cause the LDAP library to restart the `select(2)` system call when it is interrupted by the system (that is `errno` is set to `EINTR`). This option is not supported on the Macintosh and under MS-DOS.

An option can be turned off by clearing the appropriate bit in the `ld_options` field.

ERRORS

If an error occurs, these functions will return `NULL` and `errno` should be set appropriately.

ATTRIBUTES

See `attributes(5)` for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|---|
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

`ldap(3N)`, `ldap_bind(3N)`, `errno(3C)`

NOTES

There are other elements in the LDAP structure that you should not change. You should not make any assumptions about the order of elements in the LDAP structure.

| NAME | ldap_parse_result, ldap_parse_extended_result, ldap_parse_sasl_bind_result – LDAP message result parser | | | | | | |
|----------------------|--|----------------|-----------------|--------------|---|-----------------|----------|
| SYNOPSIS | <pre>cc[flag ...] file ... -lldap[library ...] #include <lber.h> #include <ldap.h> int ldap_parse_result(LDAP *ld, LDAPMessage *res, int *errcodep, char **matcheddn, char **errmsgp, char **referralsp, LDAPControl **serverctrlsp, int freeit); int ldap_parse_sasl_bind_result(LDAP *ld, LDAPMessage *res, struct berval **servercredp, int freeit); int ldap_parse_extended_result(LDAP *ld, LDAPMessage *res, char **resultoidp, struct berval **resultdata, int freeit);</pre> | | | | | | |
| DESCRIPTION | The ldap_parse_extended_result() , ldap_parse_result() and ldap_parse_sasl_bind_result() routines search for a message to parse. These functions skip messages of type LDAP_RES_SEARCH_ENTRY and LDAP_RES_SEARCH_REFERENCE . | | | | | | |
| RETURN VALUES | They return LDAP_SUCCESS if the result was successfully parsed or an LDAP error code if not (see ldap_error(3N)). | | | | | | |
| ATTRIBUTES | See attributes(5) for a description of the following attributes: | | | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>Availability</td> <td>SUNWldap (32-bit) SUNWldapx (64-bit)</td> </tr> <tr> <td>Stability Level</td> <td>Evolving</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | Availability | SUNWldap (32-bit) SUNWldapx (64-bit) | Stability Level | Evolving |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) | | | | | | |
| Stability Level | Evolving | | | | | | |
| SEE ALSO | ldap_error(3N) , ldap_result(3N) , attributes(5) | | | | | | |

| | |
|--------------------|--|
| NAME | ldap_result – wait for and return LDAP operation result |
| SYNOPSIS | <pre>cc[<i>flag...</i>] <i>file...</i> -lldap[<i>library...</i>] #include <lber.h> #include <ldap.h> int ldap_result(LDAP *ld, int msgid, int all, struct timeval *timeout, LDAPMessage **result); int ldap_msgfree(LDAPMessage *msg);</pre> |
| DESCRIPTION | <p>The ldap_result() function is used to wait for and return the result of an operation previously initiated by one of the LDAP asynchronous operation functions (for example, ldap_search(3N), ldap_modify(3N), etc.). Those functions all return <code>-1</code> in case of error, and an invocation identifier upon successful initiation of the operation. The invocation identifier is picked by the library and is guaranteed to be unique across the LDAP session. It can be used to request the result of a specific operation from ldap_result() through the <i>msgid</i> parameter.</p> <p>The ldap_result() function will block or not, depending upon the setting of the <i>timeout</i> parameter. If <i>timeout</i> is not a NULL pointer, it specifies a maximum interval to wait for the selection to complete. If <i>timeout</i> is a NULL pointer, the select blocks indefinitely. To effect a poll, the <i>timeout</i> argument should be a non-NULL pointer, pointing to a zero-valued <i>timeval</i> structure. See select(2) for further details.</p> <p>If the result of a specific operation is required, <i>msgid</i> should be set to the invocation identifier returned when the operation was initiated, otherwise <code>LDAP_RES_ANY</code> should be supplied. The <i>all</i> parameter only has meaning for search responses and is used to select whether a single entry of the search response should be returned, or all results of the search should be returned.</p> <p>A search response is made up of zero or more search entries followed by a search result. If <i>all</i> is set to <code>-</code>, search entries will be returned one at a time as they come in, via separate calls to ldap_result(). If it's set to <code>-1</code>, the search response will only be returned in its entirety, that is, after all entries and the final search result have been received.</p> <p>Upon success, the type of the result received is returned and the <i>result</i> parameter will contain the result of the operation. This result should be passed to the LDAP parsing functions, (see ldap_first_entry(3N)) for interpretation.</p> <p>The possible result types returned are:</p> <pre>#define LDAP_RES_BIND 0x61L #define LDAP_RES_SEARCH_ENTRY 0x64L</pre> |

```
#define LDAP_RES_SEARCH_RESULT 0x65L
#define LDAP_RES_MODIFY      0x67L
#define LDAP_RES_ADD         0x69L
#define LDAP_RES_DELETE      0x6bL
#define LDAP_RES_MODRDN     0x6dL
#define LDAP_RES_COMPARE     0x6fL
```

The **ldap_msgfree()** function is used to free the memory allocated for a result by **ldap_result()** or **ldap_search_s(3N)** functions. It takes a pointer to the result to be freed and returns the type of the message it freed.

ERRORS

ldap_result() returns `-1` if something bad happens, and zero if the timeout specified was exceeded.

ATTRIBUTES

See **attributes(5)** for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|---|
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

ldap(3N), **ldap_search(3N)**, **select(2)**

NOTES

This function allocates memory for results that it receives. The memory can be freed by calling **ldap_msgfree**.

| | |
|--------------------|---|
| NAME | ldap_search, ldap_search_s, ldap_search_ext, ldap_search_ext_s, ldap_search_st – LDAP search operations |
| SYNOPSIS | <pre> cc[flag ...] file ... -lldap[library ...] #include <sys/time.h> /* for struct timeval definition */ #include <lber.h> #include <ldap.h> int ldap_search(LDAP *ld, char *base, int scope, char *filter, char *attrs [], int attrsonly); int ldap_search_s(LDAP *ld, char *base, int scope, char *filter, char *attrs [], int attrsonly, LDAPMessage **res); int ldap_search_st(LDAP *ld, char *base, int scope, char *filter, char *attrs [], int attrsonly, struct timeval *timeout, LDAPMessage **res); int ldap_search_ext(LDAP *ld, char *base, int scope, char *filter, char **attrs, int attrsonly, LDAPControl **serverctrls, LDAPControl **clientctrls, struct timeval *timeoutp, int sizelimit, int *msgidp); int ldap_search_ext_s(LDAP *ld, char *base, int scope, char *filter, char **attrs, int attrsonly, LDAPControl **serverctrls, LDAPControl **clientctrls, struct timeval *timeoutp, int sizelimit); </pre> |
| DESCRIPTION | <p>These functions are used to perform LDAP search operations. ldap_search_s() does the search synchronously (that is, not returning until the operation completes). ldap_search_st() does the same, but allows a <i>timeout</i> to be specified. ldap_search() is the asynchronous version, initiating the search and returning the message id of the operation it initiated.</p> <p><i>Base</i> is the DN of the entry at which to start the search. <i>Scope</i> is the scope of the search and should be one of LDAP_SCOPE_BASE, to search the object itself, LDAP_SCOPE_ONELEVEL, to search the object's immediate children, or LDAP_SCOPE_SUBTREE, to search the object and all its descendents.</p> <p><i>Filter</i> is a string representation of the filter to apply in the search. Simple filters can be specified as <i>attributetype=attributevalue</i>. More complex filters are specified using a prefix notation according to the following BNF:</p> <pre> <filter> ::= '(' <filtercomp> ')' <filtercomp> ::= <and> <or> <not> <simple> <and> ::= '&' <filterlist> </pre> |

```

<or> ::= '|' <filterlist>
<not> ::= '!' <filter>
<filterlist> ::= <filter> | <filter> <filterlist>
<simple> ::= <attributetype> <filtertype> <attributevalue>
<filtertype> ::= '=' | '~=' | '<=' | '>='

```

The '~=' construct is used to specify approximate matching. The representation for <attributetype> and <attributevalue> are as described in RFC 1778. In addition, <attributevalue> can be a single * to achieve an attribute existence test, or can contain text and *'s interspersed to achieve substring matching.

For example, the filter "mail=*" will find any entries that have a mail attribute. The filter "mail=*@terminator.rs.itd.umich.edu" will find any entries that have a mail attribute ending in the specified string. To put parentheses in a filter, escape them with a backslash '\\' character. See RFC 1588 for a more complete description of allowable filters. See `ldap_getfilter(3N)` for functions to help in constructing search filters automatically.

Attrs is a null-terminated array of attribute types to return from entries that match *filter*. If NULL is specified, all attributes will be returned. *Attrsonly* should be set to 1 if only attribute types are wanted. It should be set to 0 if both attributes types and attribute values are wanted.

The `ldap_search_ext()` function initiates an asynchronous search operation and returns LDAP_SUCCESS if the request was successfully sent to the server, or else it returns a LDAP error code (see `ldap_error(3N)`). If successful, `ldap_search_ext()` places the message id of the request in **msgidp*. A subsequent call to `ldap_result(3N)`, can be used to obtain the result of the add request.

The `ldap_search_ext_s()` function initiates a synchronous search operation and as such returns the result of the operation itself.

ERRORS

`ldap_search_s()` and `ldap_search_st()` will return the LDAP error code resulting from the search operation. See `ldap_error(3N)` for details.

`ldap_search()` returns -1 when terminating unsuccessfully.

ATTRIBUTES

See `attributes(5)` for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|---|
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

`ldap(3N)`, `ldap_result(3N)`, `ldap_getfilter(3N)`, `ldap_error(3N)`

NOTES

Note that both read and list functionality are subsumed by these functions, by using a filter like "objectclass=*" and a scope of LDAP_SCOPE_BASE (to emulate read) or LDAP_SCOPE_ONELEVEL (to emulate list).

These functions may allocate memory which must be freed by the calling application. Return values are contained in <ldap.h> .

| | |
|--------------------|---|
| NAME | ldap_searchprefs, ldap_init_searchprefs, ldap_init_searchprefs_buf, ldap_free_searchprefs, ldap_first_searchobj, ldap_next_searchobj – LDAP search preference configuration routines |
| SYNOPSIS | <pre>cc[flag ...] file ... -lldap[library ...] # include <lber.h> # include <ldap.h> int ldap_init_searchprefs(char **file, struct ldap_searchobj **solistp); int ldap_init_searchprefs_buf(char **buf, unsigned longlen, struct ldap_searchobj **solistp); struct ldap_searchobj **ldap_free_searchprefs(struct ldap_searchobj **solist); struct ldap_searchobj **ldap_first_searchobj(struct ldap_searchobj **solist); struct ldap_searchobj **ldap_next_searchobj(struct ldap_searchobj **solist, struct ldap_searchobj **so);</pre> |
| DESCRIPTION | <p>These functions provide a standard way to access LDAP search preference configuration data. LDAP search preference configurations are typically used by LDAP client programs to specify which attributes a user may search by, labels for the attributes, and LDAP filters and scopes associated with those searches. Client software presents these choices to a user, who can then specify the type of search to be performed.</p> <p>ldap_init_searchprefs() reads a sequence of search preference configurations from a valid LDAP searchpref configuration file (see ldapsearchprefs.conf(4)). Upon success, 0 is returned and <i>solistp</i> is set to point to a list of search preference data structures.</p> <p>ldap_init_searchprefs_buf() reads a sequence of search preference configurations from <i>buf</i> (whose size is <i>buflen</i>). <i>buf</i> should point to the data in the format defined for an LDAP search preference configuration file (see ldapsearchprefs.conf(4)). Upon success, 0 is returned and <i>solistp</i> is set to point to a list of search preference data structures.</p> <p>ldap_free_searchprefs() disposes of the data structures allocated by <code>ldap_init_searchprefs()</code>.</p> |

ldap_first_searchpref() returns the first search preference data structure in the list *solist*. The *solist* is typically obtained by calling `ldap_init_searchprefs()`.

ldap_next_searchpref() returns the search preference after *so* in the template list *solist*. A NULL pointer is returned if *so* is the last entry in the list.

ERRORS

ldap_init_search_prefs() and **ldap_init_search_prefs_bufs()** return:

LDAP_SEARCHPREF_ERR_VERSION ***buf* points to data that is newer than can be handled.

LDAP_SEARCHPREF_ERR_MEM Memory allocation problem.

ATTRIBUTES

See **attributes(5)** for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|--|
| Availability | SUNWlldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

ldap(3N) , **ldapsearchprefs.conf(4)**

Yeong, W., Howes, T., and Hardcastle-Kille, S., "Lightweight Directory Access Protocol", OSI-DS-26, April 1992.

Howes, T., Hardcastle-Kille, S., Yeong, W., and Robbins, C., "Lightweight Directory Access Protocol", OSI-DS-26, April 1992.

Hardcastle-Kille, S., "A String Representation of Distinguished Names", OSI-DS-23, April 1992.

Information Processing - Open Systems Interconnection - The Directory, International Organization for Standardization. International Standard 9594, (1988).

| | |
|--------------------|--|
| NAME | ldap_sort, ldap_sort_entries, ldap_sort_values, ldap_sort_strcasecmp – LDAP entry sorting functions |
| SYNOPSIS | <pre>cc[flag ...] file ... -lldap[library ...] #include <lber.h> #include <ldap.h> ldap_sort_entries(LDAP *ld, LDAPMessage **chain, char *attr, int (*cmp)()); ldap_sort_values(LDAP *ld, char **vals, int (*cmp)()); ldap_sort_strcasecmp(char *a, char *b);</pre> |
| DESCRIPTION | <p>These functions are used to sort lists of entries and values retrieved from an LDAP server. ldap_sort_entries() is used to sort a chain of entries retrieved from an LDAP search call either by DN or by some arbitrary attribute in the entries. It takes <i>ld</i>, the LDAP structure, which is only used for error reporting, <i>chain</i>, the list of entries as returned by ldap_search_s(3N) or ldap_result(3N). <i>attr</i> is the attribute to use as a key in the sort or NULL to sort by DN, and <i>cmp</i> is the comparison function to use when comparing values (or individual DN components if sorting by DN). In this case, <i>cmp</i> should be a function taking two single values of the <i>attr</i> to sort by, and returning a value less than zero, equal to zero, or greater than zero, depending on whether the first argument is less than, equal to, or greater than the second argument. The convention is the same as used by qsort(3C), which is called to do the actual sorting.</p> <p>ldap_sort_values() is used to sort an array of values from an entry, as returned by ldap_get_values(3N). It takes the LDAP connection structure <i>ld</i>, the array of values to sort <i>vals</i>, and <i>cmp</i>, the comparison function to use during the sort. Note that <i>cmp</i> will be passed a pointer to each element in the <i>vals</i> array, so if you pass the normal char ** for this parameter, <i>cmp</i> should take two char **'s as arguments (that is, you cannot pass <i>strcasecmp</i> or its friends for <i>cmp</i>). You can, however, pass the function ldap_sort_strcasecmp() for this purpose.</p> <p>For example:</p> <pre>\011LDAP *ld; \011LDAPMessage *res; \011/* ... call to ldap_search_s(), fill in res, retrieve sn attr ... */</pre> |

```
\011/* now sort the entries on surname attribute */
\011if ( ldap_sort_entries( ld, &res, "sn", ldap_sort_strcasecmp ) != 0 )
\011\011ldap_perror( ld, "ldap_sort_entries" );
```

ATTRIBUTES

See **attributes(5)** for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|--|
| Availability | SUNWlldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

ldap(3N) , **ldap_search(3N)** , **ldap_result(3N)** , **qsort(3C)**

NOTES

The **ldap_sort_entries()** function applies the comparison function to each value of the attribute in the array as returned by a call to **ldap_get_values(3N)** , until a mismatch is found. This works fine for single-valued attributes, but may produce unexpected results for multi-valued attributes. When sorting by DN, the comparison function is applied to an exploded version of the DN, without types. The return values for all of these functions are declared in the `<ldap.h>` header file. Some functions may allocate memory which must be freed by the calling application.

| | |
|--------------------|--|
| NAME | ldap_ufn, ldap_ufn_search_s, ldap_ufn_search_c, ldap_ufn_search_ct, ldap_ufn_setfilter, ldap_ufn_setprefix, ldap_ufn_timeout – LDAP user friendly search functions |
| SYNOPSIS | <pre> cc[flag ...] file ... -lldap[library ...] #include <lber.h> #include <ldap.h> int ldap_ufn_search_c(LDAP *ld, char *ufn, char **attrs, int attrsonly, LDAPMessage **res, int (*cancelproc)(), void *cancelparm); int ldap_ufn_search_ct(LDAP *ld, char *ufn, char **attrs, int attrsonly, LDAPMessage **res, int (*cancelproc)(), void *cancelparm, char *tag1, char *tag2, char *tag3); int ldap_ufn_search_s(LDAP *ld, char *ufn, char **attrs, int attrsonly, LDAPMessage **res); LDAPFiltDesc * ldap_ufn_setfilter(LDAP *ld, char *fname); void ldap_ufn_setprefix(LDAP *ld, char *prefix); int ldap_ufn_timeout(void *tparam); </pre> |
| DESCRIPTION | <p>These functions are used to perform LDAP user friendly search operations. ldap_ufn_search_s() is the simplest form. It does the search synchronously. It takes <i>ld</i> to identify the the LDAP connection. The <i>ufn</i> parameter is the user friendly name for which to search. The <i>attrs</i> , <i>attrsonly</i> and <i>res</i> parameters are the same as for ldap_search(3N) .</p> <p>The ldap_ufn_search_c() function functions the same as ldap_ufn_search_s() , except that it takes <i>cancelproc</i> , a function to call periodically during the search. It should be a function taking a single void * argument, given by <i>cancelparm</i> . If <i>cancelproc</i> returns a non-zero result, the search will be abandoned and no results returned. The purpose of this function is to provide a way for the search to be cancelled, for example, by a user or because some other condition occurs.</p> <p>The ldap_ufn_search_ct() function is like ldap_ufn_search_c() , except that it takes three extra parameters. <i>tag1</i> is passed to the ldap_init_getfilter(3N) function when resolving the first component of the UFN. <i>tag2</i> is used when resolving intermediate components. <i>tag3</i> is used when resolving the last component. By default, the tags used by the other</p> |

UFN search functions during these three phases of the search are "ufn first", "ufn intermediate", and "ufn last".

The **ldap_ufn_setfilter()** function is used to set the `ldapfilter.conf(4)` file for use with the `ldap_init_getfilter(3N)` function to *fname*.

The **ldap_ufn_setprefix()** function is used to set the default prefix (actually, it's a suffix) appended to UFNs before searching. UFNs with fewer than three components have the prefix appended first, before searching. If that fails, the UFN is tried with progressively shorter versions of the prefix, stripping off components. If the UFN has three or more components, it is tried by itself first. If that fails, a similar process is applied with the prefix appended.

The **ldap_ufn_timeout()** function is used to set the timeout associated with **ldap_ufn_search_s()** searches. The *timeout* parameter should actually be a pointer to a struct `timeval` (this is so **ldap_ufn_timeout()** can be used as a `cancelproc` in the above functions).

ATTRIBUTES

See `attributes(5)` for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|--|
| Availability | SUNWlldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

`gettimeofday(2)`, `ldap(3N)`, `ldap_search(3N)`, `ldap_getfilter(3N)`, `ldapfilter.conf(4)`, `ldap_error(3N)`

NOTES

These functions may allocate memory. Return values are contained in `<ldap.h>`.

| | |
|--------------------|---|
| NAME | ldap_url, ldap_is_ldap_url, ldap_url_parse, ldap_free_urldesc, ldap_url_search, ldap_url_search_s, ldap_url_search_st, ldap_dns_to_url, ldap_dn_to_url – LDAP Uniform Resource Locator functions |
| SYNOPSIS | <pre> cc[flag ...] file ... -lldap[library ...] #include <lber.h> #include <ldap.h> int ldap_is_ldap_url(char *url); int ldap_url_parse(char *url, LDAPURLDesc **ludpp); ldap_free_urldesc(LDAPURLDesc *ludp); int ldap_url_search(LDAP *ld, char *url, int attrsonly); int ldap_url_search_s(LDAP *ld, char *url, int attrsonly, LDAPMessage **res); int ldap_url_search_st(LDAP *ld, char *url, int attrsonly, struct timeval *timeout, LDAPMessage **res); char * ldap_dns_to_url(LDAP *ld, char *dns_name, char *attrs, char *scope, char *filter); char * ldap_dn_to_url(LDAP *ld, char *dn, int nameparts); </pre> |
| DESCRIPTION | <p>These functions support the use of LDAP URLs (Uniform Resource Locators). LDAP URLs look like this:</p> <pre> ldap:// hostport / dn [? attributes [? scope [? filter]]] </pre> <p>where:</p> |

hostport Host name with an optional ":portnumber".

dn Base DN to be used for an LDAP search operation.

attributes Comma separated list of attributes to be retrieved.

scope One of these three strings: base one sub (default=base).

filter LDAP search filter as used in a call to `ldap_search(3N)`.

Here is an example:

```
ldap://ldap.itd.umich.edu/c=US?o,description?one?o=umich
```

URLs that are wrapped in angle-brackets and/or preceded by "URL:" are also tolerated.

ldap_is_ldap_url() returns a non-zero value if *url* looks like an LDAP URL (as opposed to some other kind of URL). It can be used as a quick check for an LDAP URL; the **ldap_url_parse()** function should be used if a more thorough check is needed.

ldap_url_parse() breaks down an LDAP URL passed in *url* into its component pieces. If successful, zero is returned, an LDAP URL description is allocated, filled in, and *ludpp* is set to point to it. See RETURN VALUES (below) for values returned upon error.

ldap_free_urldesc() should be called to free an LDAP URL description that was obtained from a call to `ldap_url_parse()`.

ldap_url_search() initiates an asynchronous LDAP search based on the contents of the *url* string. This function acts just like `ldap_search(3N)` except that many search parameters are pulled out of the URL.

ldap_url_search_s() performs a synchronous LDAP search based on the contents of the *url* string. This function acts just like `ldap_search_s(3N)` except that many search parameters are pulled out of the URL.

ldap_url_search_st() performs a synchronous LDAP URL search with a specified *timeout*. This function acts just like `ldap_search_st(3N)` except that many search parameters are pulled out of the URL.

ldap_dns_to_url() locates the LDAP URL associated with a DNS domain name. The supplied DNS domain name is converted into a distinguished name. The directory entry specified by that distinguished name is searched for a labeledURI attribute. If successful then the corresponding LDAP URL is returned. If unsuccessful then that entry's parent is searched and so on until the target distinguished name is reduced to only two nameparts. If *dns_name* is NULL then the environment variable LOCALDOMAIN is used. If *attrs* is not NULL then it is appended to the URL's attribute list. If *scope* is not NULL then

it overrides the URL's scope. If *filter* is not NULL then it is merged with the URL's filter. If an error is encountered then zero is returned, otherwise a string URL is returned. The caller should free the returned string if it is non-zero.

ldap_dn_to_url() locates the LDAP URL associated with a distinguished name. The number of nameparts in the supplied distinguished name must be provided. The specified directory entry is searched for a labeledURI attribute. If successful then the LDAP URL is returned. If unsuccessful then that entry's parent is searched and so on until the target distinguished name is reduced to only two nameparts. If an error is encountered then zero is returned, otherwise a string URL is returned. The caller should free the returned string if it is non-zero.

RETURN VALUES

Upon error, one of these values is returned for **ldap_url_parse()** :

LDAP_URL_ERR_NOTLDAP URL doesn't begin with "ldap://".

LDAP_URL_ERR_NODN URL has no DN (required).

LDAP_URL_ERR_BADSCOPE URL scope string is invalid.

LDAP_URL_ERR_MEM Can't allocate memory space.

ATTRIBUTES

See **attributes(5)** for a description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|-----------------|---|
| Availability | SUNWldap (32-bit) SUNWldapx (64-bit) |
| Stability Level | Evolving |

SEE ALSO

ldap(3N) , **ldap_search(3N)**

An LDAP URL Format , Tim Howes and Mark Smith, December 1995. Internet Draft (work in progress). Currently available at this URL:

<ftp://ds.internic.net/internet-drafts/draft-ietf-asid-ldap-format-03.txt>

| NAME | ldexp – load exponent of a floating point number | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <math.h> double ldexp(double x, int exp);</pre> | | | | |
| DESCRIPTION | The ldexp() function computes the quantity $x * 2^{exp}$. | | | | |
| RETURN VALUES | <p>Upon successful completion, ldexp() returns a <code>double</code> representing the value x multiplied by 2 raised to the power exp.</p> <p>If the value of x is NaN, NaN is returned.</p> <p>If ldexp() would cause overflow, <code>±HUGE_VAL</code> is returned (according to the sign of x), and <code>errno</code> is set to <code>ERANGE</code>.</p> <p>If ldexp() would cause underflow to 0.0, 0 is returned and <code>errno</code> may be set to <code>ERANGE</code>.</p> | | | | |
| ERRORS | <p>The ldexp() function will fail if:</p> <p>ERANGE The value to be returned would have caused overflow.</p> <p>The ldexp() function may fail if:</p> <p>ERANGE The value to be returned would have caused underflow.</p> | | | | |
| USAGE | An application wishing to check for error situations should set <code>errno</code> to 0 before calling ldexp() . If <code>errno</code> is non-zero on return, or the return value is NaN, an error has occurred. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | frexp(3C) , isnan(3M) , attributes(5) | | | | |

| | |
|--------------------|---|
| NAME | lfmt – display error message in standard format and pass to logging and monitoring services |
| SYNOPSIS | <pre>#include <pfmt.h> int lfmt(FILE *stream, long flags, char *format, ... /* arg*/);</pre> |
| DESCRIPTION | <p>The lfmt() function retrieves a format string from a locale-specific message database (unless <code>MM_NOGET</code> is specified) and uses it for <code>printf(3S)</code> style formatting of <i>args</i>. The output is displayed on <i>stream</i>. If <i>stream</i> is <code>NULL</code> no output is displayed.</p> <p>The lfmt() function encapsulates the output in the standard error message format (unless <code>MM_NOSTD</code> is specified, in which case the output is like that of printf()). It forwards its output to the logging and monitoring facility, even if <i>stream</i> is <code>NULL</code>. Optionally, lfmt() displays the output on the console with a date and time stamp.</p> <p>If the printf() format string is to be retrieved from a message database, the <i>format</i> argument must have the following structure:</p> <pre><catalog>: <msgnum>: <defmsg>.</pre> <p>If <code>MM_NOGET</code> is specified, only the <i><defmsg></i> field must be specified.</p> <p>The <i><catalog></i> field indicates the message database that contains the localized version of the format string. This field is limited to 14 characters selected from a set of all characters values, excluding the null character (<code>\0</code>) and the ASCII codes for slash (<code>/</code>) and colon (<code>:</code>).</p> <p>The <i><msgnum></i> field is a positive number that indicates the index of the string into the message database.</p> <p>If the catalog does not exist in the locale (specified by the last call to setlocale(3C) using the <code>LC_ALL</code> or <code>LC_MESSAGES</code> categories), or if the message number is out of bound, lfmt() will attempt to retrieve the message from the C locale. If this second retrieval fails, lfmt() uses the <i><defmsg></i> field of the <i>format</i> argument.</p> <p>If <i><catalog></i> is omitted, lfmt() will attempt to retrieve the string from the default catalog specified by the last call to setcat(3C). In this case, the <i>format</i> argument has the following structure:</p> <pre>: <msgnum>: <defmsg>.</pre> <p>The lfmt() function will output the message</p> <pre>Message not found!!\n</pre> |

as the format string if *<catalog>* is not a valid catalog name, if no catalog is specified (either explicitly or with `setcat()`), if *<msgnum>* is not a valid number, or if no message could be retrieved from the message databases and *<defmsg>* was omitted.

The *flags* argument determines the type of output (whether the `format` should be interpreted as it is or be encapsulated in the standard message format) and the access to message catalogs to retrieve a localized version of `format`.

The *flags* argument is composed of several groups, and can take the following values (one from each group):

Output format control

| | |
|----------|--|
| MM_NOSTD | Do not use the standard message format but interpret <code>format</code> as a <code>printf()</code> format. Only <i>catalog access control flags</i> , <i>console display control</i> and <i>logging information</i> should be specified if MM_NOSTD is used; all other flags will be ignored. |
| MM_STD | Output using the standard message format (default value is 0). |

Catalog access control

| | |
|----------|--|
| MM_NOGET | Do not retrieve a localized version of <code>format</code> . In this case, only the <i><defmsg></i> field of <code>format</code> is specified. |
| MM_GET | Retrieve a localized version of <code>format</code> from <i><catalog></i> , using <i><msgid></i> as the index and <i><defmsg></i> as the default message (default value is 0). |

Severity (standard message format only)

| | |
|------------|---|
| MM_HALT | Generate a localized version of HALT, but donot halt the machine. |
| MM_ERROR | Generate a localized version of ERROR (default value is 0). |
| MM_WARNING | Generate a localized version of WARNING. |
| MM_INFO | Generate a localized version of INFO. |

Additional severities can be defined with the `addsev(3C)` function, using number-string pairs with numeric values in the range [5-255]. The specified severity is formed by the bitwise OR operation of the numeric value and other *flags* arguments.

If the severity is not defined, `lfmt()` uses the string `SEV=N` where *N* is the integer severity value passed in *flags*.

Multiple severities passed in *flags* will not be detected as an error. Any combination of severities will be summed and the numeric value will cause the display of either a severity string (if defined) or the string `SEV=N` (if undefined).

Action

`MM_ACTION` Specify an action message. Any severity value is superseded and replaced by a localized version of `TO FIX`.

Console display control

`MM_CONSOLE` Display the message to the console in addition to the specified *stream*.

`MM_NOCONSOLE` Do not display the message to the console in addition to the specified *stream* (default value is 0).

Logging information

Major classification

Identify the source of the condition. Identifiers are: `MM_HARD` (hardware), `MM_SOFT` (software), and `MM_FIRM` (firmware).

Message source subclassification

Identify the type of software in which the problem is spotted. Identifiers are: `MM_APPL` (application), `MM_UTIL` (utility), and `MM_OPSYS` (operating system).

The `lfmt()` function displays error messages in the following format:

STANDARD ERROR MESSAGE

FORMAT
Last modified 29 Dec 1996

SunOS 5.7

1308

label: severity: text

If no *label* was defined by a call to `setlabel(3C)`, the message is displayed in the format:

severity: text

If `lfmt()` is called twice to display an error message and a helpful *action* or recovery message, the output may appear as follows:

label: severity: text
label: TO FIX: text

RETURN VALUES

Upon successful completion, `lfmt()` returns the number of bytes transmitted. Otherwise, it returns a negative value:

- 1 Write the error to *stream*.
- 2 Cannot log and/or display at console.

USAGE

Since `lfmt()` uses `gettext(3C)`, it is recommended that `lfmt()` not be used.

EXAMPLES

EXAMPLE 1 The following example

```
setlabel("UX:test");
lfmt(stderr, MM_ERROR|MM_CONSOLE|MM_SOFT|MM_UTIL,
      "test:2:Cannot open file: %s\n", strerror(errno));
```

displays the message to `stderr` and to the console and makes it available for logging:

```
UX:test: ERROR: Cannot open file: No such file or directory
```

EXAMPLE 2 The following example

```
setlabel("UX:test");
lfmt(stderr, MM_INFO|MM_SOFT|MM_UTIL,
```

```
"test:23:test facility is enabled\n");
```

displays the message to `stderr` and makes it available for logging:

```
UX:test: INFO: test facility enabled
```

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`addsev(3C)`, `gettxt(3C)`, `pfmt(3C)`, `printf(3S)`, `setcat(3C)`,
`setlabel(3C)`, `setlocale(3C)`, `attributes(5)`, `environ(5)`

| | |
|-----------------------|---|
| NAME | lgamma, lgamma_r, gamma, gamma_r - log gamma function |
| SYNOPSIS | <pre>cc [flag ...] file ... -lm [library ...] #include <math.h> extern int signgam; double lgamma(double x); double lgamma_r(double x, int * signgamp);</pre> |
| DESCRIPTION | <p>Both lgamma() and lgamma_r() return</p> $\ln \Gamma(x) $ <p>where</p> $\Gamma(x) = \int_0^{\infty} t^{x-1} e^{-t} dt$ <p>for $x > 0$ and</p> $\Gamma(x) = \pi / (\Gamma(1-x) \sin(\pi x))$ <p>for $x < 1$.</p> <p>lgamma() uses the external integer <code>signgam</code> to return the sign of $\sim(x)$ while lgamma_r() uses the user-allocated space addressed by <code>signgamp</code>.</p> |
| IDIOSYNCRASIES | <p>In the case of lgamma(), do <i>not</i> use the expression <code>signgam*exp(lgamma(x))</code> to compute</p> $g = \Gamma(x)$ <p>Instead compute lgamma() first:</p> <pre>lg = lgamma(x); g = signgam*exp(lg);</pre> <p>only after lgamma() has returned can <code>signgam</code> be correct. Note that $\sim(x)$ must overflow when x is large enough, underflow when $-x$ is large enough, and generate a division by 0 exception at the singularities x a nonpositive integer.</p> |

RETURN VALUES

For exceptional cases, **matherr**(3M) tabulates the values to be returned as dictated by various Standards.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|------------------|
| MT-Level | See NOTES below. |

SEE ALSO

matherr(3M) , **attributes**(5)

NOTES

Although **lgamma_r()** is not mentioned by POSIX.4a Draft 6, it was added to complete the functionality provided by similar thread-safe functions. This interface is subject to change to be compatible with the "spirit" of POSIX.4a when it is approved as a standard.

When compiling multi-thread applications, the **_REENTRANT** flag must be defined on the compile line. This flag should only be used in multi-thread applications.

lgamma() is unsafe in multithreaded applications. **lgamma_r()** should be used instead.

| | |
|--------------------|--|
| NAME | libdevinfo - library of device information functions |
| SYNOPSIS | <code>cc [flag...] file... -ldevinfo [library...]</code> |
| DESCRIPTION | <p>libdevinfo is a set of interfaces used to access device configuration data.</p> <p>Device configuration data is organized as a tree of device nodes, defined as <code>di_node_t</code> in the libdevinfo interfaces. Each <code>di_node_t</code> represents a physical or logical (pseudo) device. Three types of data are associated with device nodes:</p> <ul style="list-style-type: none"> ■ data defined for all device nodes (attributes) ■ properties specific to each device ■ minor node data <p>All device nodes have a set of common attributes, such as a node name, an instance number, and a driver binding name. Common device node attributes are accessed by calling interfaces listed on the <code>di_binding_name(3)</code> man page. Each device node also has a physical path, which is accessed by calling <code>di_devfs_path(3)</code>.</p> <p>Properties provide device specific information for device configuration and usage. Properties may be defined by software (<code>di_prop_t</code>) or by firmware (<code>di_prom_prop_t</code>). One way to access each <code>di_prop_t</code> is to make successive calls to <code>di_prop_next(3)</code> until <code>DI_PROP_NIL</code> is returned. For each <code>di_prop_t</code>, use interfaces on the <code>di_prop_bytes(3)</code> man page to obtain property names and values. Another way to access these properties is to call <code>di_prop_lookup_bytes(3)</code> to find the value of a property with a given name. Accessing a <code>di_prom_prop_t</code> is similar to accessing a <code>di_prop_t</code>, except that the interface names start with <code>di_prom_prop</code> and additional calls to <code>di_prom_init(3)</code> and <code>di_prom_fini(3)</code> are required.</p> <p>Minor nodes contain information exported by the device for creating special files for the device. Each device node has 0 or more minor nodes associated with it. A list minor nodes (<code>di_minor_t</code>) may be obtained by making successive calls to <code>di_minor_next(3)</code> until <code>DI_MINOR_NIL</code> is returned. For each minor node, <code>di_minor_devt(3)</code> and related interfaces are called to get minor node data.</p> <p>Using libdevinfo involves three steps:</p> <ul style="list-style-type: none"> ■ Creating a snapshot of the device tree ■ Traversing the device tree to get information of interest ■ Destroying the snapshot of the device tree |

A snapshot of the device tree is created by calling `di_init(3)` and destroyed by calling `di_fini(3)`. An application may specify the data to be included in the snapshot (full or partial tree, include or exclude properties and minor nodes) and get a handle to the root of the device tree. See `di_init(3)` for details. The application then traverses the device tree in the snapshot to obtain device configuration data.

The device tree is normally traversed through parent-child-sibling linkage. Each device node contains references to its parent, its next sibling, and the first of its children. Given the `di_node_t` returned from `di_init(3)`, one can find all children by first calling `di_child_node(3)`, followed by successive calls to `di_sibling_node(3)`, until `DI_NODE_NIL` is returned. By following this procedure recursively, an application can visit all device nodes contained in the snapshot. Two interfaces, `di_walk_node(3)` and `di_walk_minor(3)`, are provided to facilitate device tree traversal. The `di_walk_node(3)` interface visits all device nodes and executes a user-supplied callback function for each node visited. The `di_walk_minor(3)` does the same for each minor node in the device tree.

An alternative way to traverse the device tree is through the per-driver device node linkage. Device nodes contain a reference to the next device node bound to the same driver. Given the `di_node_t` returned from `di_init(3)`, an application can find all device nodes bound to a driver by first calling `di_drv_first_node(3)`, followed by successive calls to `di_drv_next_node(3)` until `DI_NODE_NIL` is returned. Note that traversing the per-driver device node list works only when the snapshot includes all device nodes.

See `libdevinfo(4)` for a complete list of `libdevinfo` interfaces. See `di_init(3)` for examples of `libdevinfo` usage. See *Writing Device Drivers* for details of Solaris device configuration.

EXAMPLES

EXAMPLE 1 Information Accessible Through `libdevinfo` Interfaces

The following example illustrates the kind of information accessible through `libdevinfo` interfaces for a device node representing a hard disk (`sd2`):

```
Attributes
node name:  sd
instance:  2
physical path:  /sbus@1f,0/espdma@e,8400000/esp@e,8800000/sd@2,0

Properties
target=2
lun=0

Minor nodes
(disk partition /dev/dsk/c0t2d0s0)
name:      a
```

```

dev_t:      0x0080010 (32/16)
spectype:   IF_BLK (block special)
(disk partition /dev/rdisk/c0t2d0s2)
name:       c,raw
dev_t:      0x0080012 (32/18)
spectype:   IF_CHR (character special)

```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---------------------|-----------------|
| MT Level | Safe |
| Interface Stability | Evolving |

SEE ALSO

devlink(1M), **prtconf(1M)**, **di_binding_name(3)**, **di_child_node(3)**, **di_devfs_path(3)**, **di_drv_first_node(3)**, **di_drv_next_node(3)**, **di_fini(3)**, **di_init(3)**, **di_minor_devt(3)**, **di_minor_next(3)**, **di_prom_fini(3)**, **di_prom_init(3)**, **di_prop_bytes(3)**, **di_prop_lookup_bytes(3)**, **di_prop_next(3)**, **di_sibling_node(3)**, **di_walk_minor(3)**, **di_walk_node(3)**, **libdevinfo(4)**, **attributes(5)**

Writing Device Drivers

NAME libthread_db – library of interfaces for monitoring and manipulating threads-related aspects of multithreaded programs

SYNOPSIS

```
cc [ flag ... ] file ... /lib/libthread_db.so.1 [ library ... ]

#include <proc_service.h>
#include <thread_db.h>

void td_event_addset(td_thr_events_t *, td_thr_events_e n);
void td_event_delset(td_thr_events_t *, td_thr_events_e n);
void td_event_emptyset(td_thr_events_t *);
void td_event_fillset(td_thr_events_t *);
void td_eventisempty(td_thr_events_t *);
void td_eventismember(td_thr_events_t *, td_thr_events_e n);

td_err_e td_init();

void td_log();

td_err_e td_sync_get_info(const td_synchandle_t *sh_p, td_syncinfo_t *si_p);
td_err_e td_sync_setstate(const td_synchandle_t *sh_p, int value);

td_err_e td_sync_waiters(const td_synchandle_t *sh_p, td_thr_iter_f *cb, void
*cb_data_p);

td_err_e td_thr_clear_event(const td_thrhandle_t *th_p, td_thr_events_t *events);

td_err_e td_ta_delete(td_thragent_t *ta_p);

td_err_e td_ta_enable_stats(const td_thragent_t *ta_p, int on_off);

td_err_e td_ta_event_addr(const td_thragent_t *ta_p, u_long event, td_notify_t
*notify_p);

td_err_e td_ta_event_getmsg(const td_thragent_t *ta_p, td_event_msg_t *msg);

td_err_e td_ta_get_nthreads(const td_thragent_t *ta_p, int *nthread_p);

td_err_e td_ta_get_ph(const td_thragent_t *ta_p, struct ps_prochandle **ph_pp);

td_err_e td_ta_get_stats(const td_thragent_t *ta_p, td_ta_stats_t *tstats);

td_err_e td_ta_map_addr2sync(const td_thragent_t *ta_p, psaddr_t addr
td_synchandle_t *sh_p);

td_err_e td_ta_map_id2thr(const td_thragent_t *ta_p, thread_t tid, td_thrhandle_t
*th_p);
```

```

td_err_e td_ta_map_lwp2thr(const td_thragent_t *ta_p, lwpid_t lwpid, td_thrhandle_t
*th_p);
td_err_e td_ta_new(struct ps_prochandle *ph_p, td_thragent_t **ta_pp);
td_err_e td_ta_reset_stats(const td_thragent_t *ta_p);
td_err_e td_ta_setconcurrency(const td_thragent_t *ta_p, int level);
td_err_e td_ta_sync_iter(const td_thragent_t *ta_p, td_sync_iter_f *cb, void
*cbdata_p);
td_err_e td_ta_thr_iter(const td_thragent_t *ta_p, td_key_iter_f *cb, void *cbdata_p);
td_err_e td_ta_tsd_iter(const td_thragent_t *ta_p, td_key_iter_f *cb, void *cbdata_p);
td_err_e td_thr_clear_event(const td_thrhandle_t *th_p, td_thr_events_t *events);
td_err_e td_thr_dbresume(const td_thrhandle_t *th_p);
td_err_e td_thr_dbsuspend(const td_thrhandle_t *th_p);
td_err_e td_thr_event_enable(const td_thrhandle_t *th_p, int on_off);
td_err_e td_thr_event_getmsg(const td_thrhandle_t, td_event_msg_t *msg);
td_err_e td_thr_get_info(const td_thrhandle_t *th_p, td_thrinfo_t *ti_p);
td_err_e td_thr_getfpregs(const td_thrhandle_t *th_p, prfpregset_t *fpregset);
td_err_e td_thr_getgregs(const td_thrhandle_t *th_p, prgregset_t regset);
td_err_e td_thr_getxregs(const td_thrhandle_t *th_p, void *xregset);
td_err_e td_thr_getxregsize(const td_thrhandle_t *th_p, int *xregsize);
td_err_e td_thr_lockowner(const td_thrhandle_t *th_p, td_sync_iter_f *cb, void
*cb_data_p);
td_err_e td_thr_set_event(const td_thrhandle_t *th_p, td_thr_events_t *events);
td_err_e td_thr_setfpregs(const td_thrhandle_t *th_p, prfpregset_t *fpregset);
td_err_e td_thr_setgregs(const td_thrhandle_t *th_p, const prgregset_t regset);
td_err_e td_thr_setprio(const td_thrhandle_t *th_p, const int new_prio);
td_err_e td_thr_setsigpending(const td_thrhandle_t *th_p, const uchar_t,
ti_pending_flag, const sigset_t ti_pending);
td_err_e td_thr_setxregs(const td_thrhandle_t *th_p, const void *xregset);
td_err_e td_thr_sigsetmask(const td_thrhandle_t *th_p, const sigset_t ti_sigmask);

```

```

td_err_e td_thr_sleepinfo(const td_thrhandle_t *th_p, td_synchandle_t *sh_p);

td_err_e td_thr_tsd(const td_thrhandle_t *th_p, const thread_key_t key, void
**data_pp);

td_err_e td_thr_validate(const td_thrhandle_t *th_p);

```

DESCRIPTION

libthread_db is a library that provides support for monitoring and manipulating threads-related aspects of a multithreaded program. There are at least two processes involved, the controlling process and one or more target processes. The controlling process is the libthread_db client, which links with libthread_db and uses libthread_db to inspect or modify threads-related aspects of one or more target processes. The target processes must be multithreaded processes that use libthread or libpthread. The controlling process may or may not be multithreaded itself.

The most commonly anticipated use for libthread_db is that the controlling process will be a debugger for a multithreaded program, hence the "db" in libthread_db.

libthread_db is dependent on the internal implementation details of libthread. It is a "friend" of libthread in the C++ sense, which is precisely the "value added" by libthread_db. It encapsulates the knowledge of libthread internals that a debugger needs in order to manipulate the threads-related state of a target process.

To be able to inspect and manipulate target processes, libthread_db makes use of certain process control primitives that must be provided by the process using libthread_db. The imported interfaces are defined in **proc_service(3T)**. In other words, the controlling process is linked with libthread_db, and it calls routines in libthread_db. libthread_db in turn calls certain routines that it expects the controlling process to provide. These process control primitives allow libthread_db to:

- Look up symbols in a target process.
- Stop and continue individual lightweight processes (LWPs) within a target process.
- Stop and continue an entire target process.
- Read and write memory and registers in a target process.

Initially, a controlling process obtains a handle for a target process. Through that handle it can then obtain handles for the component objects of the target process, its threads, its synchronization objects, and its thread-specific-data keys.

When `libthread_db` needs to return sets of handles to the controlling process, for example, when returning handles for all the threads in a target process, it uses an iterator function. An iterator function calls back a client-specified function once for each handle to be returned, passing one handle back on each call to the callback function. The calling function also passes another parameter to the iterator function, which the iterator function passes on to the callback function. This makes it easy to build a linked list of thread handles for a particular target process. The additional parameter is the head of the linked list, and the callback function simply inserts the current handle into the linked list.

Callback functions are expected to return an integer. Iteration terminates early if a callback function returns a non-zero value. Otherwise, iteration terminates when there are no more handles to pass back.

`libthread_db` relies on an "agent thread" in the target process for some of its operations. The "agent thread" is a system thread started when `libthread_db` attaches to a process through `td_ta_new(3T)`. In the current implementation, a brief window exists after the agent thread has been started, but before it has completed its initialization, in which `libthread_db` routines that require the agent thread will fail, returning a `TD_NOCAPAB` error status. This is particularly troublesome if the target process was stopped when `td_ta_new()` was called, so that the agent thread cannot be initialized. To avoid this problem, the target process must be allowed to make some forward progress after `td_ta_new()` is called. This limitation will be removed in a future release.

FUNCTIONS

| Name | Description |
|----------------------------------|---|
| <code>td_event_addset()</code> | Macro that adds a specific event type to an event set. |
| <code>td_event_delset()</code> | Macro that deletes a specific event type from an event set. |
| <code>td_event_emptyset()</code> | Macro that sets argument to NULL event set. |
| <code>td_event_fillset()</code> | Macro that sets argument to set of all events. |
| <code>td_eventisempty()</code> | Macro that tests whether an event set is the NULL set. |
| <code>td_eventismember()</code> | Macro that tests whether a specific event type is a member of an event set. |
| <code>td_init()</code> | Performs initialization for interfaces. |

| | |
|------------------------------|--|
| td_log() | Placeholder for future logging functionality. |
| td_sync_get_info() | Gets information for the synchronization object. |
| td_sync_setstate() | Sets the state of the synchronization object. |
| td_sync_waiters() | Iteration function used for return of synchronization object handles. |
| td_ta_clear_event() | Clears a set of event types in the process event mask. |
| td_ta_delete() | Deregisters target process and deallocates internal process handle. |
| td_ta_enable_stats() | Turns statistics gathering on or off for the target process. |
| td_ta_event_addr() | Returns event reporting address. |
| td_ta_event_getmsg() | Returns process event message. |
| td_ta_get_nthreads() | Gets the total number of threads in a process. . |
| td_ta_get_ph() | Returns corresponding external process handle. |
| td_ta_get_stats() | Gets statistics gathered for the target process. |
| td_ta_map_addr2sync() | Gets a synchronization object handles from a synchronization object's address. |
| td_ta_map_id2thr() | Returns a thread handle for the given thread id. |
| td_ta_map_lwp2thr() | Returns a thread handle for the given LWP id. |
| td_ta_new() | Registers target process and allocates internal process handle. |

| | |
|-------------------------------|---|
| td_ta_reset_stats() | Resets all counters for statistics gathering for the target process. |
| td_ta_setconcurrency() | Sets concurrency level for target process. |
| td_ta_set_event() | Sets a set of event types in the process event mask. |
| td_ta_sync_iter() | Returns handles of synchronization objects associated with a process. |
| td_ta_thr_iter() | Returns handles for threads that are part of the target process. |
| td_ta_tsd_iter() | Returns the thread-specific data keys in use by the current process. |
| td_thr_clear_event() | Clears a set of event types in the threads event mask. |
| td_thr_dbresume() | Resumes thread. |
| td_thr_dbsuspend() | Suspends thread. |
| td_thr_event_enable() | Enables or disables event reporting. |
| td_thr_event_getmsg() | Returns a process event message. |
| td_thr_get_info() | Gets thread information and updates |
| td_thr_getfpregs() | Gets the floating point registers for the given thread. |
| td_thr_getgregs() | Gets the general registers for a given thread. |
| td_thr_getxregs() | Gets the extra registers for the given thread. |
| td_thr_getxregsize() | Gets the size of the extra register set for the given thread. |
| td_thr_lockowner() | Iterates over the set of locks owned by a thread. <i>struct</i> . |
| td_thr_set_event() | Sets a set of event types in the threads event mask. |

| | |
|-------------------------------|---|
| td_thr_setfpregs() | Sets the floating point registers for the given thread. <i>ti_sigmask</i> |
| td_thr_setgregs() | Sets the general registers for a given thread. |
| td_thr_setprio() | Sets the priority of a thread. |
| td_thr_setsigpending() | Changes a thread's pending signal state. |
| td_thr_setxregs() | Sets the extra registers for the given thread. |
| td_thr_sigsetmask() | Sets the signal mask of the thread. |
| td_thr_sleepinfo() | Returns the synchronization handle for the object on which a thread is blocked. |
| td_thr_tsd() | Gets a thread's thread-specific data. |
| td_thr_validate() | Tests a thread handle for validity. |

FILES

/usr/lib/libthread_db.so.1

ATTRIBUTES

See `attributes(5)` for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`libthread(3T)`, `proc_service(3T)`, `td_event_addset(3T)`,
`td_event_delset(3T)`, `td_event_emptyset(3T)`,
`td_event_fillset(3T)`, `td_eventisempty(3T)`, `td_eventismember(3T)`,
`td_init(3T)`, `td_log(3T)`, `td_sync_get_info(3T)`,
`td_sync_waiters(3T)`, `td_ta_delete(3T)`, `td_ta_enable_stats(3T)`,
`td_ta_event_addr(3T)`, `td_ta_event_getmsg(3T)`,
`td_ta_get_nthreads(3T)`, `td_ta_get_ph(3T)`, `td_ta_get_stats(3T)`,
`td_ta_map_addr2sync(3T)`, `td_ta_map_id2thr(3T)`,
`td_ta_map_lwp2thr(3T)`, `td_ta_new(3T)`, `td_ta_reset_stats(3T)`,
`td_ta_set_event(3T)`, `td_ta_setconcurrency(3T)`,
`td_ta_sync_iter(3T)`, `td_ta_thr_iter(3T)`, `td_ta_tsd_iter(3T)`,
`td_thr_clear_event(3T)`, `td_thr_dbresume(3T)`,

```
td_thr_dbsuspend(3T), td_thr_event_enable(3T),  
td_thr_event_getmsg(3T), td_thr_get_info(3T),  
td_thr_getfpregs(3T), td_thr_getxregs(3T),  
td_thr_getxregsize(3T), td_thr_lockowner(3T),  
td_thr_set_event(3T), td_thr_setfpregs(3T), td_thr_setgregs(3T),  
td_thr_setprio(3T), td_thr_setsigmask(3T),  
td_thr_setsigpending(3T), td_thr_setxregs(3T),  
td_thr_sleepinfo(3T), td_thr_tsd(3T), td_thr_validate(3T),  
thr_getspecific(3T), libthread(4), libthread_db(4), attributes(5)
```

| | |
|--------------------|--|
| NAME | libtnfctl – library for TNF probe control in a process or the kernel |
| SYNOPSIS | <pre>#include <tnf/tnfctl.h> cc [flag ...] file ... -ltnfctl [library ...] (void);</pre> |
| DESCRIPTION | <p>libtnfctl is a library that provides an API to control TNF ("Trace Normal Form") probes within a process or the kernel. See tracing(3X) for an overview of the Solaris tracing architecture. The client of libtnfctl controls probes in one of four modes:</p> <p>internal mode The target is the controlling process itself; that is, the client controls its own probes.</p> <p>direct mode The target is a separate process; a client can either exec(2) a program or attach to a running process for probe control. libtnfctl uses proc(4) on the target process for probe and process control in this mode, and additionally provides basic process control features.</p> <p>indirect mode The target is a separate process, but the controlling process is already using proc(4) to control the target, and hence libtnfctl cannot use those interfaces directly. Use this mode to control probes from within a debugger. In this mode, the client must provide a set of functions that libtnfctl can use to query and update the target process.</p> <p>kernel mode The target is the Solaris kernel. A process is controlled "externally" if it is being controlled in either direct mode or indirect mode. Alternatively, a process is controlled "internally" when it uses internal mode to control its own probes.</p> <p>There can be only one client at a time doing probe control on a given process. Therefore, it is not possible for a process to be controlled internally while it is being controlled externally. It is also not possible to have a process controlled by multiple external processes. Similarly, there can be only one process at a time doing kernel probe control. Note, however, that while a given target may only be controlled by one libtnfctl client, a single client may control an arbitrary number of targets. That is, it is possible for a process to simultaneously control its own probes, probes in other processes, and probes in the kernel.</p> |

The following tables denotes the modes applicable to all `libtnfctl` interfaces (INT = internal mode; D = direct mode; IND = indirect mode; K = kernel mode).

These interfaces create handles in the specified modes:

| | | | | |
|-------------------------------------|-----|---|-----|---|
| <code>tnfctl_internal_open()</code> | INT | | | |
| <code>tnfctl_exec_open()</code> | | D | | |
| <code>tnfctl_pid_open()</code> | | D | | |
| <code>tnfctl_indirect_open()</code> | | | IND | |
| <code>tnfctl_kernel_open()</code> | | | | K |

These interfaces are used with the specified modes:

| | | | | |
|---|-----|---|-----|---|
| <code>tnfctl_continue()</code> | | D | | |
| <code>tnfctl_probe_connect()</code> | INT | D | IND | |
| <code>tnfctl_probe_connect_all()</code> | INT | D | IND | |
| <code>tnfctl_trace_attrs_get()</code> | INT | D | IND | K |
| <code>tnfctl_buffer_loc()</code> | INT | D | IND | K |
| <code>tnfctl_register_funcs()</code> | INT | D | IND | K |
| <code>tnfctl_probe_apply()</code> | INT | D | IND | K |
| <code>tnfctl_probe_apply_ids()</code> | INT | D | IND | K |
| <code>tnfctl_probe_attr_get()</code> | INT | D | IND | K |
| <code>tnfctl_probe_enable()</code> | INT | D | IND | K |
| <code>tnfctl_probe_disable()</code> | INT | D | IND | K |
| <code>tnfctl_probe_trace()</code> | INT | D | IND | K |
| <code>tnfctl_check_attrs()</code> | INT | D | IND | K |
| <code>tnfctl_close()</code> | INT | D | IND | K |
| <code>tnfctl_strerror()</code> | INT | D | IND | K |
| <code>tnfctl_buffer_dealloc()</code> | | | | K |
| <code>tnfctl_trace_state_set()</code> | | | | K |
| <code>tnfctl_filter_state_set()</code> | | | | K |

| | | | |
|-----------------------------|--|--|---|
| tnfctl_filter_list_get() | | | K |
| tnfctl_filter_list_add() | | | K |
| tnfctl_filter_list_delete() | | | K |

When using `libtnfctl`, the first task is to create a handle for controlling probes. Function `tnfctl_internal_open()` creates an internal mode handle for controlling probes in the same process, as described above. Functions `tnfctl_pid_open()` and `tnfctl_exec_open()` create handles in direct mode. `tnfctl_indirect_open()` creates an indirect mode handle, and `tnfctl_kernel_open()` creates a kernel mode handle. A handle is required for use in nearly all other `libtnfctl` functions. `tnfctl_close()` releases the resources associated with a handle.

`tnfctl_continue()` is used in direct mode to resume execution of the target process.

`tnfctl_buffer_alloc()` allocates a trace file or, in kernel mode, a trace buffer.

`tnfctl_probe_apply()` and `tnfctl_probe_apply_ids()` call a specified function for each probe or for a designated set of probes.

`tnfctl_register_funcs()` registers functions to be called whenever new probes are seen or probes have disappeared, providing an opportunity to do one-time processing for each probe.

`tnfctl_check_libs()` is used primarily in indirect mode to check whether any new probes have appeared, that is, they have been made available by `dlopen(3X)`, or have disappeared, that is, they have disassociated from the process by `dlclose(3X)`.

`tnfctl_probe_enable()` and `tnfctl_probe_disable()` control whether the probe, when hit, will be ignored.

`tnfctl_probe_trace()` and `tnfctl_probe_untrace()` control whether an enabled probe, when hit, will cause an entry to be made in the trace file.

`tnfctl_probe_connect()` and `tnfctl_probe_disconnect_all()` control which functions, if any, are called when an enabled probe is hit.

`tnfctl_probe_state_get()` returns information about the status of a probe, such as whether it is currently enabled.

`tnfctl_trace_attrs_get()` returns information about the tracing session, such as the size of the trace buffer or trace file.

`tnfctl_strerror()` maps a `tnfctl` error code to a string, for reporting purposes.

The remaining interfaces apply only to kernel mode.

tnfctl_trace_state_set() controls the master switch for kernel tracing. See **prex(1)** for more details.

tnfctl_filter_state_set(), **tnfctl_filter_list_get()**, **tnfctl_filter_list_add()**, and **tnfctl_filter_list_delete()** allow a set of processes to be specified for which probes will not be ignored when hit. This prevents kernel activity caused by uninteresting processes from cluttering up the kernel's trace buffer.

tnfctl_buffer_dealloc() deallocates the kernel's internal trace buffer.

RETURN VALUES

TNFCTL_ERR_NONE is returned upon success.

ERRORS

The error codes for libtnfctl are:

| | |
|--------------------------|---|
| TNFCTL_ERR_ACCES | Permission denied. |
| TNFCTL_ERR_NOTTARGET | The target process completed. |
| TNFCTL_ERR_ALLOCFAIL | A memory allocation failure occurred. |
| TNFCTL_ERR_INTERNAL | An internal error occurred. |
| TNFCTL_ERR_SIZETOOSMALL | The requested trace size is too small. |
| TNFCTL_ERR_SIZETOOBIG | The requested trace size is too big. |
| TNFCTL_ERR_BADARG | Bad input argument. |
| TNFCTL_ERR_NOTDYNAMIC | The target is not a dynamic executable. |
| TNFCTL_ERR_NOLIBTNFPROBE | libtnfprobe.so not linked in target. |
| TNFCTL_ERR_BUFBROKEN | Tracing is broken in the target. |
| TNFCTL_ERR_BUFEXISTS | A buffer already exists. |
| TNFCTL_ERR_NOBUF | No buffer exists. |
| TNFCTL_ERR_BADDEALLOC | Cannot deallocate buffer. |
| TNFCTL_ERR_NOPROCESS | No such target process exists. |
| TNFCTL_ERR_FILENOTFOUND | File not found. |
| TNFCTL_ERR_BUSY | Cannot attach to process or kernel because it is already tracing. |

| | |
|-------------------------|-------------------------------|
| TNFCTL_ERR_INVALIDPROBE | Probe no longer valid. |
| TNFCTL_ERR_USR1 | Error code reserved for user. |
| TNFCTL_ERR_USR2 | Error code reserved for user. |
| TNFCTL_ERR_USR3 | Error code reserved for user. |
| TNFCTL_ERR_USR4 | Error code reserved for user. |
| TNFCTL_ERR_USR5 | Error code reserved for user. |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| Availability | SUNWtnfc |
| MT Level | MT-Safe with exceptions |

SEE ALSO

prex(1), **exec(2)**, **dldclose(3X)**, **dlopen(3X)**, **TNF_PROBE(3X)**, **tnfctl_buffer_alloc(3X)**, **tnfctl_buffer_dealloc(3X)**, **tnfctl_check_libs(3X)**, **tnfctl_close(3X)**, **tnfctl_continue(3X)**, **tnfctl_internal_open(3X)**, **tnfctl_exec_open(3X)**, **tnfctl_filter_list_add(3X)**, **tnfctl_filter_list_delete(3X)**, **tnfctl_filter_list_get(3X)**, **tnfctl_filter_state_set(3X)**, **tnfctl_kernel_open(3X)**, **tnfctl_pid_open(3X)**, **tnfctl_probe_apply(3X)**, **tnfctl_probe_apply_ids(3X)**, **tnfctl_probe_connect(3X)**, **tnfctl_probe_disable(3X)**, **tnfctl_probe_enable(3X)**, **tnfctl_probe_state_get(3X)**, **tnfctl_probe_trace(3X)**, **tnfctl_probe_untrace(3X)**, **tnfctl_indirect_open(3X)**, **tnfctl_register_funcs(3X)**, **tnfctl_strerror(3X)**, **tnfctl_trace_attrs_get(3X)**, **tnfctl_trace_state_set(3X)**, **libtnfctl(4)**, **proc(4)**, **attributes(5)**

Programming Utilities Guide Linker and Libraries Guide

NOTES

This API is MT-Safe. Multiple threads may concurrently operate on independent **tnfctl** handles, which is the typical behavior expected. **libtnfctl** does not support multiple threads operating on the same **tnfctl** handle. If this is desired, it is the client's responsibility to implement locking to ensure that two threads that use the same **tnfctl** handle are not simultaneously in a **libtnfctl** interface.

| | |
|--------------------|--|
| NAME | lio_listio – list directed I/O |
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <aio.h> int lio_listio(int mode, struct aiocb * const list[], int nent, struct sigevent *sig);</pre> |
| DESCRIPTION | <p>The lio_listio() function allows the calling process, LWP, or thread, to initiate a list of I/O requests within a single function call.</p> <p>The <i>mode</i> argument takes one of the values LIO_WAIT or LIO_NOWAIT declared in <aio.h> and determines whether the function returns when the I/O operations have been completed, or as soon as the operations have been queued. If the <i>mode</i> argument is LIO_WAIT, the function waits until all I/O is complete and the <i>sig</i> argument is ignored.</p> <p>If the <i>mode</i> argument is LIO_NOWAIT, the function returns immediately, and asynchronous notification occurs, according to the <i>sig</i> argument, when all the I/O operations complete. If <i>sig</i> is NULL, or the <i>sigev_signo</i> member of the <i>sigevent</i> structure referenced by <i>sig</i> is zero, then no asynchronous notification occurs. If <i>sig</i> is not NULL, asynchronous notification occurs when all the requests in <i>list</i> have completed. If <i>sig->sigev_notify</i> is SIGEV_NONE, then no signal will be posted upon I/O completion, but the error status and the return status for the operation will be set appropriately. If <i>sig->sigev_notify</i> is SIGEV_SIGNAL, then the signal specified in <i>sig->sigev_signo</i> will be sent to the process. If the SA_SIGINFO flag is set for that signal number, then the signal will be queued to the process and the value specified in <i>sig->sigev_value</i> will be the <i>si_value</i> component of the generated signal (see siginfo(5)).</p> <p>The <i>list</i> argument is an array of pointers to <i>aiocb</i> structures. The array contains <i>nent</i> elements. The array may contain null elements, which are ignored.</p> <p>The <i>aio_lio_opcode</i> field of each <i>aiocb</i> structure specifies the operation to be performed. The supported operations are LIO_READ, LIO_WRITE, and LIO_NOP; these symbols are defined in <aio.h>. The LIO_NOP operation causes the list entry to be ignored. If the <i>aio_lio_opcode</i> element is equal to LIO_READ, then an I/O operation is submitted as if by a call to aio_read(3R) with the <i>aiocbp</i> equal to the address of the <i>aiocb</i> structure. If the <i>aio_lio_opcode</i> element is equal to LIO_WRITE, then an I/O operation is submitted as if by a call to aio_write(3R) with the <i>aiocbp</i> equal to the address of the <i>aiocb</i> structure.</p> <p>The <i>aio_fildes</i> member specifies the file descriptor on which the operation is to be performed.</p> |

The *aio_buf* member specifies the address of the buffer to or from which the data is to be transferred.

The *aio_nbytes* member specifies the number of bytes of data to be transferred.

The members of the *aio_cb* structure further describe the I/O operation to be performed, in a manner identical to that of the corresponding *aio_cb* structure when used by the `aio_read(3R)` and `aio_write(3R)` functions.

The *nent* argument specifies how many elements are members of the list, that is, the length of the array.

The behavior of this function is altered according to the definitions of synchronized I/O data integrity completion and synchronized I/O file integrity completion if synchronized I/O is enabled on the file associated with *aio_fildes*. (see `fcntl(5)` definitions of `O_DSYNC` and `O_SYNC`.)

For regular files, no data transfer will occur past the offset maximum established in the open file description associated with *aio_cb->aio_fildes*.

RETURN VALUES

If the *mode* argument has the value `LIO_NOWAIT`, and the I/O operations are successfully queued, `lio_listio()` returns 0; otherwise, it returns -1, and sets `errno` to indicate the error.

If the *mode* argument has the value `LIO_WAIT`, and all the indicated I/O has completed successfully, `lio_listio()` returns 0; otherwise, it returns -1, and sets `errno` to indicate the error.

In either case, the return value only indicates the success or failure of the `lio_listio()` call itself, not the status of the individual I/O requests. In some cases, one or more of the I/O requests contained in the list may fail. Failure of an individual request does not prevent completion of any other individual request. To determine the outcome of each I/O request, the application must examine the error status associated with each *aio_cb* control block. Each error status so returned is identical to that returned as a result of an `aio_read(3R)` or `aio_write(3R)` function.

ERRORS

The `lio_listio()` function will fail if:

- | | |
|---------------|--|
| EAGAIN | The resources necessary to queue all the I/O requests were not available. The error status for each request is recorded in the <code>aio_error</code> member of the corresponding <i>aio_cb</i> structure, and can be retrieved using <code>aio_error(3R)</code> . |
| EAGAIN | The number of entries indicated by <i>nent</i> would cause the system-wide limit <code>AIO_MAX</code> to be exceeded. |
| EINVAL | The <i>mode</i> argument is an improper value, or the value of <i>nent</i> is greater than <code>AIO_LISTIO_MAX</code> . |

- EINTR** A signal was delivered while waiting for all I/O requests to complete during an `LIO_WAIT` operation. Note that, since each I/O operation invoked by `lio_listio()` may possibly provoke a signal when it completes, this error return may be caused by the completion of one (or more) of the very I/O operations being awaited. Outstanding I/O requests are not canceled, and the application can use `aio_fsync(3R)` to determine if any request was initiated; `aio_return(3R)` to determine if any request has completed; or `aio_error(3R)` to determine if any request was canceled.
- EIO** One or more of the individual I/O operations failed. The application can use `aio_error(3R)` to check the error status for each `aio_cb` structure to determine the individual request(s) that failed.
- ENOSYS** The `lio_listio()` function is not supported by the system. In addition to the errors returned by the `lio_listio()` function, if the `lio_listio()` function succeeds or fails with errors of `EAGAIN`, `EINTR`, or `EIO`, then some of the I/O specified by the list may have been initiated. If the `lio_listio()` function fails with an error code other than `EAGAIN`, `EINTR`, or `EIO`, no operations from the list have been initiated. The I/O operation indicated by each list element can encounter errors specific to the individual read or write function being performed. In this event, the error status for each `aio_cb` control block contains the associated error code. The error codes that can be set are the same as would be set by a `read(2)` or `write(2)` function, with the following additional error codes possible:
- EAGAIN** The requested I/O operation was not queued due to resource limitations.
- ECANCELED** The requested I/O was canceled before the I/O completed due to an explicit `aio_cancel(3R)` request.
- EFBIG** The `aio_cb->aio_lio_opcode` is `LIO_WRITE`, the file is a regular file, `aio_cb->aio_nbytes` is greater than 0, and the `aio_cb->aio_offset` is greater than or equal to the offset maximum in the open file description associated with `aio_cb->aio_fildes`.
- EINPROGRESS** The requested I/O is in progress.
- EOVERFLOW** The `aio_cb->aio_lio_opcode` is `LIO_READ`, the file is a regular file, `aio_cb->aio_nbytes` is greater than 0, and the `aio_cb->aio_offset` is before the end-of-file and is greater

than or equal to the offset maximum in the open file description associated with *aiochp->aio_fildes*.

USAGE The **lio_listio()** function has a transitional interface for 64-bit file offsets. See **lf64(5)**.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **close(2)**, **exec(2)**, **exit(2)**, **fork(2)**, **lseek(2)**, **read(2)**, **write(2)**, **aio_cancel(3R)**, **aio_fsync(3R)**, **aio_read(3R)**, **aio_return(3R)**, **attributes(5)**, **aio(5)**, **fcntl(5)**, **lf64(5)**, **siginfo(5)**, **signal(5)**

NOTES Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned **-1** and set **errno** to **ENOSYS**.

| | |
|----------------------|--|
| NAME | listen – listen for socket connections and limit the queue of incoming connections |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lxnet [<i>library</i> ...] #include <sys/socket.h></pre> <pre>int listen(int <i>socket</i>, int <i>backlog</i>);</pre> |
| DESCRIPTION | <p>The listen() function marks a connection-mode socket, specified by the <i>socket</i> argument, as accepting connections, and limits the number of outstanding connections in the socket's listen queue to the value specified by the <i>backlog</i> argument.</p> <p>If listen() is called with a <i>backlog</i> argument value that is less than 0, the function sets the length of the socket's listen queue to 0.</p> <p>The implementation may include incomplete connections in the queue subject to the queue limit. The implementation may also increase the specified queue limit internally if it includes such incomplete connections in the queue subject to this limit.</p> <p>Implementations may limit the length of the socket's listen queue. If <i>backlog</i> exceeds the implementation-dependent maximum queue length, the length of the socket's listen queue will be set to the maximum supported value.</p> <p>The socket in use may require the process to have appropriate privileges to use the listen() function.</p> |
| RETURN VALUES | Upon successful completions, listen() returns 0. Otherwise, -1 is returned and <i>errno</i> is set to indicate the error. |
| ERRORS | <p>The listen() function will fail if:</p> <p>EBADF The <i>socket</i> argument is not a valid file descriptor.</p> <p>EDESTADDRREQ The socket is not bound to a local address, and the protocol does not support listening on an unbound socket.</p> <p>EINVAL The <i>socket</i> is already connected.</p> <p>ENOTSOCK The <i>socket</i> argument does not refer to a socket.</p> <p>EOPNOTSUPP The socket protocol does not support listen().</p> <p>The listen() function may fail if:</p> <p>EACCES The calling process does not have the appropriate privileges.</p> |

EINVALThe *socket* has been shut down.**ENOBUFS**

Insufficient resources are available in the system to complete the call.

ATTRIBUTESSee **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO**accept(3XN)**, **connect(3XN)**, **socket(3XN)**, **attributes(5)**

| | |
|--------------------|---|
| NAME | localeconv – get numeric formatting information |
| SYNOPSIS | <pre>#include <locale.h> struct lconv *localeconv(void);</pre> |
| DESCRIPTION | <p>The localeconv() function sets the components of an object with type <code>struct lconv</code> (defined in <code><locale.h></code>) with the values appropriate for the formatting of numeric quantities (monetary and otherwise) according to the rules of the current locale (see setlocale(3C)). The definition of <code>struct lconv</code> is given below (the values for the fields in the “C” locale are given in comments).</p> <pre>char *decimal_point; /* "." */ char *thousands_sep; /* "" (zero length string) */ char *grouping; /* "" */ char *int_curr_symbol; /* "" */ char *currency_symbol; /* "" */ char *mon_decimal_point; /* "" */ char *mon_thousands_sep; /* "" */ char *mon_grouping; /* "" */ char *positive_sign; /* "" */ char *negative_sign; /* "" */ char int_frac_digits; /* CHAR_MAX */ char frac_digits; /* CHAR_MAX */ char p_cs_precedes; /* CHAR_MAX */ char p_sep_by_space; /* CHAR_MAX */ char n_cs_precedes; /* CHAR_MAX */ char n_sep_by_space; /* CHAR_MAX */ char p_sign_posn; /* CHAR_MAX */ char n_sign_posn; /* CHAR_MAX */</pre> <p>The members of the structure with type <code>char *</code> are strings, any of which (except <code>decimal_point</code>) can point to a null string (<code>""</code>), to indicate that the value is not available in the current locale or is of zero length. The members with type <code>char</code> are non-negative numbers, any of which can be <code>CHAR_MAX</code> (defined in the <code><limits.h></code> header) to indicate that the value is not available in the current locale. The members are the following:</p> <pre>char *decimal_point</pre> <p>The decimal-point character used to format non-monetary quantities.</p> <pre>char *thousands_sep</pre> <p>The character used to separate groups of digits to the left of the decimal-point character in formatted non-monetary quantities.</p> |

```
char *grouping
```

A string in which each element is taken as an integer that indicates the number of digits that comprise the current group in a formatted non-monetary quantity. The elements of `grouping` are interpreted according to the following:

- | | |
|-----------------------|---|
| <code>CHAR_MAX</code> | No further grouping is to be performed. |
| <code>0</code> | The previous element is to be repeatedly used for the remainder of the digits. |
| <i>other</i> | The value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits to the left of the current group. |

```
char *int_curr_symbol
```

The international currency symbol applicable to the current locale, left-justified within a four-character space-padded field. The character sequences should match with those specified in *ISO 4217 Codes for the Representation of Currency and Funds*.

```
char *currency_symbol
```

The local currency symbol applicable to the current locale.

```
char *mon_decimal_point
```

The decimal point used to format monetary quantities.

```
char *mon_thousands_sep
```

The separator for groups of digits to the left of the decimal point in formatted monetary quantities.

```
char *mon_grouping
```

A string in which each element is taken as an integer that indicates the number of digits that comprise the current group in a formatted monetary quantity. The elements of `mon_grouping` are interpreted according to the rules described under `grouping`.

char *positive_sign

The string used to indicate a non-negative-valued formatted monetary quantity.

char *negative_sign

The string used to indicate a negative-valued formatted monetary quantity.

char int_frac_digits

The number of fractional digits (those to the right of the decimal point) to be displayed in an internationally formatted monetary quantity.

char frac_digits

The number of fractional digits (those to the right of the decimal point) to be displayed in a formatted monetary quantity.

char p_cs_precedes

Set to 1 or 0 if the `currency_symbol` respectively precedes or succeeds the value for a non-negative formatted monetary quantity.

char p_sep_by_space

Set to 1 or 0 if the `currency_symbol` respectively is or is not separated by a space from the value for a non-negative formatted monetary quantity.

char n_cs_precedes

Set to 1 or 0 if the `currency_symbol` respectively precedes or succeeds the value for a negative formatted monetary quantity.

char n_sep_by_space

Set to 1 or 0 if the `currency_symbol` respectively is or is not separated by a space from the value for a negative formatted monetary quantity.

char p_sign_posn

Set to a value indicating the positioning of the `positive_sign` for a non-negative formatted monetary quantity. The value of `p_sign_posn` is interpreted according to the following:

- 0 Parentheses surround the quantity and `currency_symbol`.
- 1 The sign string precedes the quantity and `currency_symbol`.
- 2 The sign string succeeds the quantity and `currency_symbol`.
- 3 The sign string immediately precedes the `currency_symbol`.
- 4 The sign string immediately succeeds the `currency_symbol`.

`char n_sign_posn`

Set to a value indicating the positioning of the `negative_sign` for a negative formatted monetary quantity. The value of `n_sign_posn` is interpreted according to the rules described under `p_sign_posn`.

RETURN VALUES

The `localeconv()` function returns a pointer to the filled-in object. The structure pointed to by the return value may be overwritten by a subsequent call to `localeconv()`.

USAGE

The `localeconv()` function can be used safely in multithreaded applications, as long as `setlocale(3C)` is not being called to change the locale.

EXAMPLES

EXAMPLE 1 Rules used by four countries to format monetary quantities.

The following table illustrates the rules used by four countries to format monetary quantities.

| Country | Positive format | Negative format | International format |
|-------------|-----------------|-----------------|----------------------|
| Italy | L.1.234 | -1.1.234 | ITL.1.234 |
| Netherlands | F 1.234,56 | F -1.234,56 | NLG 1.234,56 |
| Norway | kr1.234,56 | kr1.234,56- | NOK 1.234,56 |
| Switzerland | SFrs.1,234.56 | SFrs.1,234.56C | CHF 1,234.56 |

For these four countries, the respective values for the monetary members of the structure returned by `localeconv()` are as follows:

| | Italy | Netherlands | Norway | Switzerland |
|-------------------|--------|-------------|--------|-------------|
| int_curr_symbol | "ITL." | "NLG " | "NOK " | "CHF " |
| currency_symbol | "L." | "F" | "kr" | "Sfrs." |
| mon_decimal_point | "" | "," | "," | ." |
| mon_thousands_sep | ." | ." | ." | ." |
| mon_grouping | "\3" | "\3" | "\3" | "\3" |
| positive_sign | "" | "" | "" | "" |
| negative_sign | "-" | "-" | "-" | "C" |
| int_frac_digits | 0 | 2 | 2 | 2 |
| frac_digits | 0 | 2 | 2 | 2 |
| p_cs_precedes | 1 | 1 | 1 | 1 |
| p_sep_by_space | 0 | 1 | 0 | 0 |
| n_cs_precedes | 1 | 1 | 1 | 1 |
| n_sep_by_space | 0 | 1 | 0 | 0 |
| p_sign_posn | 1 | 1 | 1 | 1 |
| n_sign_posn | 1 | 4 | 2 | 2 |

FILES

/usr/lib/locale/*locale*/LC_MONETARY/monetary

LC_MONETARY database for *locale*

/usr/lib/locale/*locale*/LC_NUMERIC/numeric

LC_NUMERIC database for *locale*

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |
| CSI | Enabled |

SEE ALSO

setlocale(3C), **attributes(5)**, **environ(5)**

| | |
|--------------------|--|
| NAME | lockf – record locking on files |
| SYNOPSIS | <pre>#include <unistd.h> int lockf(int <i>fdes</i>, int <i>function</i>, off_t <i>size</i>);</pre> |
| DESCRIPTION | <p>The lockf() function allows sections of a file to be locked; advisory or mandatory write locks depending on the mode bits of the file (see chmod(2)). Locking calls from other processes that attempt to lock the locked file section will either return an error value or be put to sleep until the resource becomes unlocked. All the locks for a process are removed when the process terminates. See fcntl(2) for more information about record locking.</p> <p>The <i>fdes</i> argument is an open file descriptor. The file descriptor must have O_WRONLY or O_RDWR permission in order to establish locks with this function call.</p> <p>The <i>function</i> argument is a control value that specifies the action to be taken. The permissible values for <i>function</i> are defined in <unistd.h> as follows:</p> <pre>#define F_ULOCK 0 /* unlock previously locked section */ #define F_LOCK 1 /* lock section for exclusive use */ #define F_TLOCK 2 /* test & lock section for exclusive use */ #define F_TEST 3 /* test section for other locks */</pre> <p>All other values of <i>function</i> are reserved for future extensions and will result in an error if not implemented.</p> <p>F_TEST is used to detect if a lock by another process is present on the specified section. F_LOCK and F_TLOCK both lock a section of a file if the section is available. F_ULOCK removes locks from a section of the file.</p> <p>The <i>size</i> argument is the number of contiguous bytes to be locked or unlocked. The resource to be locked or unlocked starts at the current offset in the file and extends forward for a positive size and backward for a negative size (the preceding bytes up to but not including the current offset). If <i>size</i> is zero, the section from the current offset through the largest file offset is locked (that is, from the current offset through the present or any future end-of-file). An area need not be allocated to the file in order to be locked as such locks may exist past the end-of-file.</p> <p>The sections locked with F_LOCK or F_TLOCK may, in whole or in part, contain or be contained by a previously locked section for the same process. Locked sections will be unlocked starting at the the point of the offset through <i>size</i> bytes or to the end of file if <i>size</i> is (off_t) 0. When this situation occurs, or if this situation occurs in adjacent sections, the sections are combined into a single section. If the request requires that a new element be</p> |

added to the table of active locks and this table is already full, an error is returned, and the new section is not locked.

`F_LOCK` and `F_TLOCK` requests differ only by the action taken if the resource is not available. `F_LOCK` will cause the calling process to sleep until the resource is available. `F_TLOCK` will cause the function to return a `-1` and set `errno` to `EAGAIN` if the section is already locked by another process.

File locks are released on first close by the locking process of any file descriptor for the file.

`F_ULOCK` requests may, in whole or in part, release one or more locked sections controlled by the process. When sections are not fully released, the remaining sections are still locked by the process. Releasing the center section of a locked section requires an additional element in the table of active locks. If this table is full, an `errno` is set to `EDEADLK` and the requested section is not released.

An `F_ULOCK` request in which `size` is non-zero and the offset of the last byte of the requested section is the maximum value for an object of type `off_t`, when the process has an existing lock in which `size` is 0 and which includes the last byte of the requested section, will be treated as a request to unlock from the start of the requested section with a `size` equal to 0. Otherwise, an `F_ULOCK` request will attempt to unlock only the requested section.

A potential for deadlock occurs if a process controlling a locked resource is put to sleep by requesting another process's locked resource. Thus calls to `lockf()` or `fcntl(2)` scan for a deadlock prior to sleeping on a locked resource. An error return is made if sleeping on the locked resource would cause a deadlock.

Sleeping on a resource is interrupted with any signal. The `alarm(2)` function may be used to provide a timeout facility in applications that require this facility.

RETURN VALUES

Upon successful completion, 0 is returned. Otherwise, `-1` is returned and `errno` is set to indicate the error.

ERRORS

The `lockf()` function will fail if:

| | |
|-------------------------|---|
| EBADF | The <i>files</i> argument is not a valid open file descriptor; or function is <code>F_LOCK</code> or <code>F_TLOCK</code> and <i>files</i> is not a valid file descriptor open for writing. |
| EACCES or EAGAIN | The function argument is <code>F_TLOCK</code> or <code>F_TEST</code> and the section is already locked by another process. |
| EDEADLK | The function argument is <code>F_LOCK</code> and a deadlock is detected. |

| | |
|--|--|
| EINTR | A signal was caught during execution of the function. |
| ECOMM | The <i>files</i> argument is on a remote machine and the link to that machine is no longer active. |
| EINVAL | The function argument is not one of <code>F_LOCK</code> , <code>F_TLOCK</code> , <code>F_TEST</code> , or <code>F_ULOCK</code> ; or <i>size</i> plus the current file offset is less than 0. |
| E_OVERFLOW | The offset of the first, or if <i>size</i> is not 0 then the last, byte in the requested section cannot be represented correctly in an object of type <code>off_t</code> . |
| The <code>lockf()</code> function may fail if: | |
| EAGAIN | The function argument is <code>F_LOCK</code> or <code>F_TLOCK</code> and the file is mapped with <code>mmap(2)</code> . |
| EDEADLK or ENOLCK | The function argument is <code>F_LOCK</code> , <code>F_TLOCK</code> , or <code>F_ULOCK</code> , and the request would cause the number of locks to exceed a system-imposed limit. |
| EOPNOTSUPP or EINVAL | The locking of files of the type indicated by the <i>files</i> argument is not supported. |

USAGE

Record-locking should not be used in combination with the `fopen(3S)`, `fread(3S)`, `fwrite(3S)` and other `stdio` functions. Instead, the more primitive, non-buffered functions (such as `open(2)`) should be used. Unexpected results may occur in processes that do buffering in the user address space. The process may later read/write data which is/was locked. The `stdio` functions are the most common source of unexpected buffering.

The `alarm(2)` function may be used to provide a timeout facility in applications requiring it.

The `lockf()` function has a transitional interface for 64-bit file offsets. See `1f64(5)`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`intro(2)`, `alarm(2)`, `chmod(2)`, `close(2)`, `creat(2)`, `fcntl(2)`, `mmap(2)`,
`open(2)`, `read(2)`, `write(2)`, `attributes(5)`, `lf64(5)`

| NAME | log10 – base 10 logarithm function | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | cc [<i>flag ...</i>] <i>file ...</i> -lm [<i>library ...</i>] #include <math.h> double log10(double x); | | | | |
| DESCRIPTION | The log10() function computes the base 10 logarithm of x , $\log_{10}(x)$. The value of x must be positive. | | | | |
| RETURN VALUES | Upon successful completion, log10() returns the base 10 logarithm of x . If x is NaN, NaN is returned. If x is less than 0, <code>-HUGE_VAL</code> or NaN is returned, and <code>errno</code> is set to EDOM. If x is 0, <code>-HUGE_VAL</code> is returned and <code>errno</code> may be set to ERANGE. For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4. | | | | |
| ERRORS | The log10() function will fail if: EDOM The value of x is negative. The log10() function may fail if: ERANGE The value of x is 0. No other errors will occur. | | | | |
| USAGE | An application wishing to check for error situations should set <code>errno</code> to 0 before calling log10() . If <code>errno</code> is non-zero on return, or the return value is NaN, an error has occurred. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | isnan(3M) , log(3M) , matherr(3M) , pow(3M) , attributes(5) , standards(5) | | | | |

NAME | log1p – compute natural logarithm

SYNOPSIS | cc [*flag ...*] *file ...* -lm [*library ...*]
 | #include <math.h>
 |
 | double log1p(double x);

DESCRIPTION | The **log1p()** function computes $\log_e(1.0 + x)$. The value of *x* must be greater than -1.0 .

RETURN VALUES | Upon successful completion, **log1p()** returns the natural logarithm of $1.0 + x$.
 | If *x* is NaN, **log1p()** returns NaN.
 | If *x* is less than -1.0 , **log1p()** returns `-HUGE_VAL` or NaN and sets `errno` to EDOM.
 | If *x* is -1.0 , **log1p()** returns `-HUGE_VAL` and may set `errno` to ERANGE.
 | For exceptional cases, **matherr(3M)** tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS | The **log1p()** function will fail if:
 | **EDOM** The value of *x* is less than -1.0 .
 | The **log1p()** function may fail and set `errno` to:
 | **ERANGE** the value of *x* is -1.0 .

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **log(3M)**, **matherr(3M)**, **attributes(5)**, **standards(5)**

| NAME | log – natural logarithm function | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | cc [<i>flag ...</i>] <i>file ...</i> -lm [<i>library ...</i>] #include <math.h> double log (double <i>x</i>); | | | | |
| DESCRIPTION | The log() function computes the natural logarithm of <i>x</i> , $\log_e(x)$. The value of <i>x</i> must be positive. | | | | |
| RETURN VALUES | Upon successful completion, log() returns the natural logarithm of <i>x</i> . If <i>x</i> is NaN, NaN is returned. If <i>x</i> is less than 0, -HUGE_VAL or NaN is returned and <i>errno</i> is set to EDOM. If <i>x</i> is 0, -HUGE_VAL is returned and <i>errno</i> may be set to ERANGE. In IEEE 754 mode (the <code>-Xlibmieee</code> cc compilation option), if <i>x</i> is Inf or a quiet NaN, <i>x</i> is returned; if <i>x</i> is a signaling NaN, a quiet NaN is returned and the invalid operation exception is raised; if <i>x</i> is 1, 0 is returned; for all other positive <i>x</i> , a normalized number is returned and the inexact exception is raised. For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4. | | | | |
| ERRORS | The log() function will fail if: EDOM The value of <i>x</i> is negative. The log() function may fail if: ERANGE the value of <i>x</i> is 0. No other errors will occur. | | | | |
| USAGE | An application wishing to check for error situations should set <i>errno</i> to 0 before calling log() . If <i>errno</i> is non-zero on return, or the return value is NaN, an error has occurred. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | exp(3M) , isnan(3M) , log10(3M) , log1p(3M) , matherr(3M) , attributes(5) , standards(5) | | | | |

NAME | logb – radix-independent exponent

SYNOPSIS | cc [*flag ...*] *file ...* -lm [*library ...*]
 | #include <math.h>
 |
 | double logb(double x);

DESCRIPTION | The **logb()** function computes the exponent of x , which is the integral part of $\log_r |x|$, as a signed floating point value, for non-zero x , where r is the radix of the machine's floating-point arithmetic.

RETURN VALUES | Upon successful completion, **logb()** returns the exponent of x .
 | If x is 0.0, **logb()** returns `-HUGE_VAL` and sets `errno` to `EDOM`.
 | If x is $\pm\text{Inf}$, **logb()** returns `+Inf`.
 | If x is NaN, **logb()** returns NaN.
 | For exceptional cases, **matherr(3M)** tabulates the values to be returned as dictated by various Standards.

ERRORS | The **logb()** function will fail if:
 | **EDOM** The x argument is 0.0.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **ilogb(3M)**, **matherr(3M)**, **attributes(5)**

| | |
|----------------------|---|
| NAME | <code>_longjmp</code> , <code>_setjmp</code> – non-local goto |
| SYNOPSIS | <pre>#include <setjmp.h> void _longjmp(jmp_buf env, int val); int _setjmp(jmp_buf env);</pre> |
| DESCRIPTION | <p>The <code>_longjmp()</code> and <code>_setjmp()</code> functions are identical to <code>longjmp(3C)</code> and <code>setjmp(3C)</code>, respectively, with the additional restriction that <code>_longjmp()</code> and <code>_setjmp()</code> do not manipulate the signal mask.</p> <p>If <code>_longjmp()</code> is called even though <code>env</code> was never initialized by a call to <code>_setjmp()</code>, or when the last such call was in a function that has since returned, the results are undefined.</p> |
| RETURN VALUES | Refer to <code>longjmp(3C)</code> and <code>setjmp(3C)</code> . |
| ERRORS | No errors are defined. |
| USAGE | <p>If <code>_longjmp()</code> is executed and the environment in which <code>_setjmp()</code> was executed no longer exists, errors can occur. The conditions under which the environment of the <code>_setjmp()</code> no longer exists include exiting the function that contains the <code>_setjmp()</code> call, and exiting an inner block with temporary storage. This condition might not be detectable, in which case the <code>_longjmp()</code> occurs and, if the environment no longer exists, the contents of the temporary storage of an inner block are unpredictable. This condition might also cause unexpected process termination. If the function has returned, the results are undefined.</p> <p>Passing <code>longjmp()</code> a pointer to a buffer not created by <code>setjmp()</code>, passing <code>_longjmp()</code> a pointer to a buffer not created by <code>_setjmp()</code>, passing <code>siglongjmp(3C)</code> a pointer to a buffer not created by <code>sigsetjmp(3C)</code> or passing any of these three functions a buffer that has been modified by the user can cause all the problems listed above, and more.</p> <p>The <code>_longjmp()</code> and <code>_setjmp()</code> functions are included to support programs written to historical system interfaces. New applications should use <code>siglongjmp(3C)</code> and <code>sigsetjmp(3C)</code> respectively.</p> |
| SEE ALSO | <code>longjmp(3C)</code> , <code>setjmp(3C)</code> , <code>siglongjmp(3C)</code> , <code>sigsetjmp(3C)</code> |

| | |
|----------------------|--|
| NAME | longname – return full terminal type name |
| SYNOPSIS | <pre>#include <curses.h> const char *longname(void);</pre> |
| DESCRIPTION | The longname() function returns a pointer to a static area containing a verbose description (128 characters or fewer) of the terminal. The area is defined after calls to initscr(3XC) , newterm(3XC) , or setupterm(3XC) . The value should be saved if longname() is going to be used with multiple terminals since it will be overwritten with a new value after each call to newterm() or setupterm() . |
| RETURN VALUES | On success, the longname() function returns a pointer to a verbose description of the terminal. Otherwise, it returns a null pointer. |
| ERRORS | None. |
| SEE ALSO | initscr(3XC) , newterm(3XC) , setupterm(3XC) |

| | |
|----------------------|---|
| NAME | lsearch, lfind – linear search and update |
| SYNOPSIS | <pre>#include <search.h> void * lsearch(const void * key, void * base, size_t * nelp, size_t width, int (* compar))(const void *, const void *); void * lfind(const void * key, const void * base, size_t * nelp, size_t width, int (* compar))(const void *, const void *);</pre> |
| DESCRIPTION | <p>The lsearch() function is a linear search routine generalized from Knuth (6.1) Algorithm S. (See <i>The Art of Computer Programming, Volume 3, Section 6.1, by Donald E. Knuth.</i>) It returns a pointer into a table indicating where a datum may be found. If the datum does not occur, it is added at the end of the table. The <i>key</i> argument points to the datum to be sought in the table. The <i>base</i> argument points to the first element in the table. The <i>nelp</i> argument points to an integer containing the current number of elements in the table. The integer is incremented if the datum is added to the table. The <i>width</i> argument is the size of an element in bytes. The <i>compar</i> argument is a pointer to the comparison function that the user must supply (strcmp(3C) for example). It is called with two arguments that point to the elements being compared. The function must return zero if the elements are equal and non-zero otherwise.</p> <p>The lfind() function is the same as lsearch() except that if the datum is not found, it is not added to the table. Instead, a null pointer is returned.</p> <p>It is important to note the following:</p> <ul style="list-style-type: none"> ■ the pointers to the key and the element at the base of the table may be pointers to any type. ■ The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared. ■ The value returned should be cast into type pointer-to-element. |
| RETURN VALUES | If the searched-for datum is found, both lsearch() and lfind() return a pointer to it. Otherwise, lfind() returns <code>NULL</code> and lsearch() returns a pointer to the newly added element. |
| USAGE | Undefined results can occur if there is not enough room in the table to add a new item. |
| EXAMPLES | <p>EXAMPLE 1 A sample code using the lsearch() function.</p> <p>This program will read in less than <code>TABSIZE</code> strings of length less than <code>ELSIZE</code> and store them in a table, eliminating duplicates, and then will print each entry.</p> |

```

#include <search.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

#define TABSIZE 50
#define ELSIZE 120

main()
{
    char line[ELSIZE];           /* buffer to hold input string */
    char tab[TABSIZE][ELSIZE];  /* table of strings */
    size_t nel = 0;
    /* number of entries in tab */
    int i;

    while (fgets(line, ELSIZE, stdin) != NULL &&
           nel < TABSIZE)
        (void) lsearch(line, tab, &nel, ELSIZE, mycmp);
    for( i = 0; i < nel; i++ )
        (void)fputs(tab[i], stdout);
    return 0;
}

```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

bsearch(3C), **hsearch(3C)**, **string(3C)**, **tsearch(3C)**, **attributes(5)**

The Art of Computer Programming, Volume 3, Sorting and Searching by Donald E. Knuth, published by Addison-Wesley Publishing Company, 1973.

| | |
|--------------------|--|
| NAME | madvise – provide advice to VM system |
| SYNOPSIS | <pre>#include <sys/types.h> #include <sys/mman.h> int madvise(caddr_t <i>addr</i>, size_t <i>len</i>, int <i>advice</i>);</pre> |
| DESCRIPTION | <p>madvise() advises the kernel that a region of user mapped memory in the range [<i>addr</i>, <i>addr + len</i>) will be accessed following a type of pattern. The kernel uses this information to optimize the procedure for manipulating and maintaining the resources associated with the specified mapping range.</p> <p>Values for <i>advice</i> are defined in <code><sys/mman.h></code> as:</p> <pre>#define MADV_NORMAL 0x0 /* No further special treatment */ #define MADV_RANDOM 0x1 /* Expect random page references */ #define MADV_SEQUENTIAL 0x2 /* Expect sequential page references */ #define MADV_WILLNEED 0x3 /* Will need these pages */ #define MADV_DONTNEED 0x4 /* Don't need these pages */</pre> <p>MADV_NORMAL The default system characteristic where accessing memory within the address range causes the system to read data from the mapped file. The kernel reads all data from files into pages which are retained for a period of time as a “cache.” System pages can be a scarce resource, so the kernel steals pages from other mappings when needed. This is a likely occurrence, but adversely affects system performance only if a large amount of memory is accessed.</p> <p>MADV_RANDOM Tells the kernel to read in a minimum amount of data from a mapped file on any single particular access. If MADV_NORMAL is in effect when an address of a mapped file is accessed, the system tries to read in as much data from the file as reasonable, in anticipation of other accesses within a certain locality.</p> <p>MADV_SEQUENTIAL Tells the system that addresses in this range are likely to be accessed only once, so the system will free the resources mapping the address range as quickly as possible. This is used in the cat(1) and cp(1) utilities.</p> |

MADV_WILLNEED Tells the system that a certain address range is definitely needed so the kernel will start reading the specified range into memory. This can benefit programs wanting to minimize the time needed to access memory the first time, as the kernel would need to read in from the file.

MADV_DONTNEED Tells the kernel that the specified address range is no longer needed, so the system starts to free the resources associated with the address range.

madvise() should be used by programs with specific knowledge of their access patterns over a memory object, such as a mapped file, to increase system performance.

RETURN VALUES

madvise() returns:

0 on success.

-1 on failure and sets `errno` to indicate the error.

ERRORS

EINVAL *addr* is not a multiple of the page size as returned by `sysconf(3C)`.

The length of the specified address range is less than or equal to 0, or the advice was invalid.

EIO An I/O error occurred while reading from or writing to the file system.

ENOMEM Addresses in the range [*addr*, *addr + len*) are outside the valid range for the address space of a process, or specify one or more pages that are not mapped.

ESTALE Stale nfs file handle.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`cat(1)`, `cp(1)`, `mmap(2)`, `sysconf(3C)`, `attributes(5)`

| | |
|----------------------|--|
| NAME | maillock, mailunlock, touchlock – functions to manage lockfile(s) for user's mailbox |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lmail [<i>library</i> ...] #include <maillock.h> int maillock(const char * <i>user</i>, int <i>retrycnt</i>); void mailunlock(void); void touchlock(void);</pre> |
| DESCRIPTION | <p>The maillock() function attempts to create a lockfile for the user's mailfile. If a lockfile already exists, and it has not been modified in the last 5 minutes, maillock() will remove the lockfile and set its own lockfile.</p> <p>It is crucial that programs locking mail files refresh their locks at least every three minutes to maintain the lock. Refresh the lockfile by calling the routine touchlock() with no arguments.</p> <p>The algorithm used to determine the age of the lockfile takes into account clock drift between machines using a network file system. A zero is written into the lockfile so that the lock will be respected by systems running the standard version of System V.</p> <p>If the lockfile has been modified in the last 5 minutes the process will sleep until the lock is available. The sleep algorithm is to sleep for 5 seconds times the attempt number. That is, the first sleep will be for 5 seconds, the next sleep will be for 10 seconds, etc. until the number of attempts reaches <i>retrycnt</i> .</p> <p>When the lockfile is no longer needed, it should be removed by calling mailunlock() .</p> <p><i>user</i> is the login name of the user for whose mailbox the lockfile will be created. maillock() assumes that user's mailfiles are in the "standard" place as defined in <maillock.h> .</p> |
| RETURN VALUES | The following return code definitions are contained in <maillock.h> . |

| | | | |
|---------|------------|---|--------------------------------------|
| #define | L_SUCCESS | 0 | /* Lockfile created or removed */ |
| #define | L_NAMELEN | 1 | /* Recipient name > 13 chars */ |
| #define | L_TMPLOCK | 2 | /* Can't create tmp file */ |
| #define | L_TMPWRITE | 3 | /* Can't write pid into lockfile */ |
| #define | L_MAXTRYS | 4 | /* Failed after retrycnt attempts */ |
| #define | L_ERROR | 5 | /* Check errno for reason */ |

FILES

LIBDIR/l1ib-mail.ln
 LIBDIR/mail.a
 /var/mail/*
 /var/mail/*.lock

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

attributes(5)

NOTES

mailunlock() will only remove the lockfile created from the most previous call to **maillock()**. Calling **maillock()** for different users without intervening calls to **mailunlock()** will cause the initially created lockfile(s) to remain, potentially blocking subsequent message delivery until the current process finally terminates.

| NAME | makecontext, swapcontext – manipulate user contexts | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <ucontext.h> void makecontext(ucontext_t * ucp, void(* func)(), int argc, ...); int swapcontext(ucontext_t * oucp, const ucontext_t * ucp);</pre> | | | | |
| DESCRIPTION | <p>These functions are useful for implementing user-level context switching between multiple threads of control within a process.</p> <p>The makecontext() function modifies the context specified by <i>ucp</i>, which has been initialized using getcontext(); when this context is resumed using swapcontext() or setcontext() (see getcontext(2)), program execution continues by calling the function <i>func</i>, passing it the arguments that follow <i>argc</i> in the makecontext() call. The integer value of <i>argc</i> must be one-greater-than the number of arguments that follow <i>argc</i>; otherwise, the behavior is undefined. For 5 arguments, the value of <i>argc</i> must be 6.</p> <p>Before a call is made to makecontext(), the context being modified should have a stack allocated for it. The value of <i>argc</i> must match the number of integer arguments passed to func(), otherwise the behavior is undefined.</p> <p>The <i>uc_link</i> member is used to determine the context that will be resumed when the context being modified by makecontext() returns. The <i>uc_link</i> member should be initialized prior to the call to makecontext().</p> <p>The swapcontext() function saves the current context in the context structure pointed to by <i>oucp</i> and sets the context to the context structure pointed to by <i>ucp</i>.</p> | | | | |
| RETURN VALUES | On successful completion, swapcontext() returns 0. Otherwise, -1 is returned and <i>errno</i> is set to indicate the error. | | | | |
| ERRORS | <p>The makecontext() and swapcontext() functions will fail if:</p> <p>EFAULT The <i>ucp</i> or <i>oucp</i> argument points to an invalid address.</p> <p>ENOMEM The <i>ucp</i> argument does not have enough stack left to complete the operation.</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |

SEE ALSO `exit(2)`, `getcontext(2)`, `sigaction(2)`, `sigprocmask(2)`,
`attributes(5)`, `ucontext(5)`

NOTES The size of the `ucontext_t` structure may change in future releases. To remain binary compatible, users of these features must always use **`makecontext()`** or **`getcontext()`** to create new instances of them.

| NAME | makedev, major, minor – manage a device number | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <sys/types.h> #include <sys/mkdev.h> dev_t makedev(major_t maj, minor_t min); major_t major(dev_t device); minor_t minor(dev_t device);</pre> | | | | |
| DESCRIPTION | <p>The makedev() function returns a formatted device number on success and NODEV on failure. The <i>maj</i> argument is the major number. The <i>min</i> argument is the minor number. The makedev() function can be used to create a device number for input to mknod(2) .</p> <p>The major() function returns the major number component from <i>device</i> .</p> <p>The minor() function returns the minor number component from <i>device</i> .</p> | | | | |
| RETURN VALUES | Upon successful completion, makedev() returns a formatted device number. Otherwise, NODEV is returned and errno is set to indicate the error. | | | | |
| ERRORS | <p>The makedev() function will fail if:</p> <p>EINVAL One or both of the arguments <i>maj</i> and <i>min</i> is too large, or the <i>device</i> number created from <i>maj</i> and <i>min</i> is NODEV .</p> <p>The major() function will fail if:</p> <p>EINVAL The <i>device</i> argument is NODEV , or the major number component of <i>device</i> is too large.</p> <p>The minor() function will fail if:</p> <p>EINVAL The <i>device</i> argument is NODEV .</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | mknod(2) , stat(2) , attributes(5) | | | | |

| | |
|--------------------|---|
| NAME | malloc, calloc, free, memalign, realloc, valloc, alloca – memory allocator |
| SYNOPSIS | <pre>#include <stdlib.h> void * malloc(size_t size); void * calloc(size_t nelem, size_t elsize); void free(void * ptr); void * memalign(size_t alignment, size_t size); void * realloc(void * ptr, size_t size); void * valloc(size_t size); #include <alloca.h> void * alloca(size_t size);</pre> |
| DESCRIPTION | <p>The malloc() and free() functions provide a simple, general-purpose memory allocation package. The malloc() function returns a pointer to a block of at least <code>size</code> bytes suitably aligned for any use.</p> <p>The argument to free() is a pointer to a block previously allocated by malloc(), calloc(), or realloc(). This space is made available for further allocation after free() is executed. If <code>ptr</code> is a null pointer, no action occurs.</p> <p>Undefined results will occur if the space assigned by malloc() is overrun or if some random number is passed to free().</p> <p>The calloc() function allocates space for an array of <code>nelem</code> elements of size <code>elsize</code>. The space is initialized to zeros.</p> <p>The memalign() function allocates <code>size</code> bytes on a specified alignment boundary and returns a pointer to the allocated block. The value of the returned address is guaranteed to be an even multiple of <code>alignment</code>. Note that the value of <code>alignment</code> must be a power of two, and must be greater than or equal to the size of a word.</p> <p>The realloc() function changes the size of the block pointed to by <code>ptr</code> to <code>size</code> bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes. If <code>ptr</code> is <code>NULL</code>, realloc() behaves like malloc() for the specified size. If <code>size</code> is 0 and <code>ptr</code> is not a null pointer, the object pointed to is freed.</p> <p>The valloc() function is equivalent to <code>memalign(sysconf(_SC_PAGESIZE), size)</code>.</p> <p>Each of the allocation functions returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.</p> |

The **malloc()** , **realloc()** , **memalign()** , and **valloc()** functions will fail if there is not enough available memory.

The **alloca()** function allocates *size* bytes of space in the stack frame of the caller, and returns a pointer to the allocated block. This temporary space is automatically freed when the caller returns. If the allocated block is beyond the current stack limit, the resulting behavior is undefined.

RETURN VALUES

If there is no available memory, **malloc()** , **realloc()** , **memalign()** , **valloc()** , and **calloc()** return a null pointer. When **realloc()** returns `NULL` , the block pointed to by *ptr* is left intact. If *size* , *nelem* , or *elsize* is 0 , a unique pointer to the arena is returned.

If **malloc()** , **calloc()** , or **realloc()** returns unsuccessfully, `errno` will be set to indicate the error. The **free()** function does not set `errno` .

ERRORS

The **malloc()** , **calloc()** , and **realloc()** functions will fail if:

- ENOMEM** The physical limits of the system are exceeded by *size* bytes of memory which cannot be allocated.
- EAGAIN** There is not enough memory available *at this point in time* to allocate *size* bytes of memory; but the application could try again later.

USAGE

Comparative features of **malloc(3C)** , **bsdmalloc(3X)** , and **malloc(3X)** are as follows:

- The **bsdmalloc(3X)** routines afford better performance, but are space-inefficient.
- The **malloc(3X)** routines are space-efficient, but have slower performance.
- The standard, fully SCD-compliant `malloc` routines are a trade-off between performance and space-efficiency.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

brk(2) , **getrlimit(2)** , **bsdmalloc(3X)** , **malloc(3X)** , **mapmalloc(3X)** , **watchmalloc(3X)** , **attributes(5)**

WARNINGS

Undefined results will occur if the size requested for a block of memory exceeds the maximum size of a process's heap, which may be obtained with `getrlimit(2)`

The `alloca()` function is machine-, compiler-, and most of all, system-dependent. Its use is strongly discouraged.

| | |
|--------------------|--|
| NAME | malloc, free, realloc, calloc, malloc, mallinfo – memory allocator |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lmalloc [<i>library</i> ...] #include <stdlib.h> void * malloc(size_t <i>size</i>); void free(void * <i>ptr</i>); void * realloc(void * <i>ptr</i>, size_t <i>size</i>); void * calloc(size_t <i>nelem</i>, size_t <i>elsize</i>); #include <malloc.h> int malloc(int <i>cmd</i>, int <i>value</i>); struct mallinfo mallinfo(void);</pre> |
| DESCRIPTION | <p>malloc() and free() provide a simple general-purpose memory allocation package.</p> <p>malloc() returns a pointer to a block of at least <i>size</i> bytes suitably aligned for any use.</p> <p>The argument to free() is a pointer to a block previously allocated by malloc(); after free() is performed this space is made available for further allocation, and its contents have been destroyed (but see malloc() below for a way to change this behavior). If <i>ptr</i> is a null pointer, no action occurs.</p> <p>Undefined results occur if the space assigned by malloc() is overrun or if some random number is handed to free().</p> <p>realloc() changes the size of the block pointed to by <i>ptr</i> to <i>size</i> bytes and returns a pointer to the (possibly moved) block. The contents are unchanged up to the lesser of the new and old sizes. If <i>ptr</i> is a null pointer, realloc() behaves like malloc() for the specified size. If <i>size</i> is zero and <i>ptr</i> is not a null pointer, the object it points to is freed.</p> |

calloc() allocates space for an array of *nelem* elements of size *elsize* . The space is initialized to zeros.

mallopt() provides for control over the allocation algorithm. The available values for *cmd* are:

| | |
|----------|---|
| M_MXFAST | Set <i>maxfast</i> to <i>value</i> . The algorithm allocates all blocks below the size of <i>maxfast</i> in large groups and then does them out very quickly. The default value for <i>maxfast</i> is 24. |
| M_NLBLKS | Set <i>numlblks</i> to <i>value</i> . The above mentioned “large groups” each contain <i>numlblks</i> blocks. <i>numlblks</i> must be greater than 0. The default value for <i>numlblks</i> is 100. |
| M_GRAIN | Set <i>grain</i> to <i>value</i> . The sizes of all blocks smaller than <i>maxfast</i> are considered to be rounded up to the nearest multiple of <i>grain</i> . <i>grain</i> must be greater than 0. The default value of <i>grain</i> is the smallest number of bytes that will allow alignment of any data type. Value will be rounded up to a multiple of the default when <i>grain</i> is set. |
| M_KEEP | Preserve data in a freed block until the next malloc() , realloc() , or calloc() . This option is provided only for compatibility with the old version of malloc() , and it is not recommended. |

These values are defined in the `<malloc.h>` header.

mallopt() may be called repeatedly, but may not be called after the first small block is allocated.

mallinfo() provides instrumentation describing space usage. It returns the `mallinfo` structure with the following members:

```

unsigned long arena;\011\011/* total space in arena */
unsigned long ordblks;\011\011/* number of ordinary blocks */
unsigned long smlbks;\011\011/* number of small blocks */
unsigned long hblkhd;\011\011/* space in holding block headers */
unsigned long hblks;\011\011/* number of holding blocks */
unsigned long usmlbks;\011/* space in small blocks in use */
unsigned long fsmblks;\011/* space in free small blocks */
unsigned long uordblks;\011/* space in ordinary blocks in use */
unsigned long fordblks;\011/* space in free ordinary blocks */
unsigned long keepcost;\011/* space penalty if keep option */
\011\011\011\011/* is used */

```

The `mallinfo` structure is defined in the `<malloc.h>` header.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

RETURN VALUES

malloc() , **realloc()** , and **calloc()** return a `NULL` pointer if there is not enough available memory. When **realloc()** returns `NULL` , the block pointed to by *ptr* is left intact. If **mallopt()** is called after any allocation or if *cmd* or *value* are invalid, non-zero is returned. Otherwise, it returns zero.

ERRORS

If **malloc()** , **calloc()** , or **realloc()** returns unsuccessfully, `errno` will be set to indicate the following:

ENOMEM *size* bytes of memory exceeds the physical limits of your system, and cannot be allocated.

EAGAIN There is not enough memory available at this point in time to allocate *size* bytes of memory; but the application could try again later.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

brk(2) , **bsdmalloc(3X)** , **mapmalloc(3X)** , **watchmalloc(3X)** , **malloc(3C)** , **mtmalloc(3T)** , **libmtmalloc(4)** , **attributes(5)**

NOTES

Note that unlike **malloc(3C)** , this package does not preserve the contents of a block when it is freed, unless the `M_KEEP` option of **mallopt()** is used.

Undocumented features of **malloc(3C)** have not been duplicated.

Function prototypes for **malloc()** , **realloc()** , **calloc()** , and **free()** are also defined in the `<malloc.h>` header for compatibility with old applications. New applications should include `<stdlib.h>` to access the prototypes for these functions. Comparative Features of **malloc(3X)** , **bsdmalloc(3X)** , and **malloc(3C)**

- These **malloc(3X)** routines are space-efficient, but have slower performance.
- The **bsdmalloc(3X)** routines afford better performance, but are space-inefficient.
- The standard, fully SCD-compliant **malloc(3C)** routines are a trade-off between performance and space-efficiency.

free() does not set `errno` .

| | |
|--------------------|--|
| NAME | mapmalloc – memory allocator |
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lmapmalloc [<i>library ...</i>] #include <stdlib.h> void *malloc(size_t size); void *calloc(size_t nelem, size_t elsize); void free(void * ptr); void *realloc(void *ptr, size_t size);</pre> |
| DESCRIPTION | <p>The collection of <code>malloc</code> routines in this library use <code>mmap(2)</code> instead of <code>sbrk(2)</code> for acquiring new heap space. The routines in this library are intended to be used only if necessary, when applications must call <code>sbrk()</code>, but need to call other library routines that might call <code>malloc</code>. The algorithms used by these routines are not sophisticated. There is no reclaiming of memory.</p> <p><code>malloc()</code> and <code>free()</code> provide a simple general-purpose memory allocation package.</p> <p><code>malloc()</code> returns a pointer to a block of at least <code>size</code> bytes suitably aligned for any use.</p> <p>The argument to <code>free()</code> is a pointer to a block previously allocated by <code>malloc()</code>, <code>calloc()</code> or <code>realloc()</code>. If <code>ptr</code> is a <code>NULL</code> pointer, no action occurs.</p> <p>Undefined results will occur if the space assigned by <code>malloc()</code> is overrun or if some random number is handed to <code>free()</code>.</p> <p><code>calloc()</code> allocates space for an array of <code>nelem</code> elements of size <code>elsize</code>. The space is initialized to zeros.</p> <p><code>realloc()</code> changes the size of the block pointed to by <code>ptr</code> to <code>size</code> bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes. If <code>ptr</code> is <code>NULL</code>, <code>realloc()</code> behaves like <code>malloc()</code> for the specified size. If <code>size</code> is zero and <code>ptr</code> is not a null pointer, the object pointed to is freed.</p> <p>Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.</p> <p><code>malloc()</code> and <code>realloc()</code> will fail if there is not enough available memory.</p> <p>Entry points for <code>malloc_debug()</code>, <code>mallocmap()</code>, <code>mallopt()</code>, <code>mallinfo()</code>, <code>memalign()</code>, and <code>valloc()</code>, are empty routines, and are provided only to protect the user from mixing <code>malloc()</code> functions from different implementations.</p> |

RETURN VALUES

If there is no available memory, **malloc()**, **realloc()**, and **calloc()** return a null pointer. When **realloc()** returns `NULL`, the block pointed to by *ptr* is left intact. If *size*, *nelem*, or *elsize* is 0, a unique pointer to the arena is returned.

FILES

`/usr/lib/libmapmalloc`

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

brk(2), **getrlimit(2)**, **mmap(2)**, **realloc(3C)**, **malloc(3X)**, **attributes(5)**

| | | | | | | | | | | | | | |
|--------------------|--|--------|---------------------------|------|----------------------|----------|--------------------------|-----------|---------------------------|-------|----------------------------|-------|------------------------------|
| NAME | matherr – math library exception-handling function | | | | | | | | | | | | |
| SYNOPSIS | <pre>#include <math.h> int matherr(struct exception *exc);</pre> | | | | | | | | | | | | |
| DESCRIPTION | <p>The The System V Interface Definition, Third Edition (SVID3) specifies that certain <code>libm</code> functions call <code>matherr()</code> when exceptions are detected. Users may define their own mechanisms for handling exceptions, by including a function named <code>matherr()</code> in their programs. The <code>matherr()</code> function is of the form described above. When an exception occurs, a pointer to the exception structure <code>exc</code> will be passed to the user-supplied <code>matherr()</code> function. This structure, which is defined in the <code><math.h></code> header file, is as follows:</p> <pre>struct exception { int type; char *name; double arg1, arg2, retval; };</pre> <p>The <code>type</code> member is an integer describing the type of exception that has occurred, from the following list of constants (defined in the header file):</p> <table border="0"> <tr> <td style="padding-right: 20px;">DOMAIN</td> <td>argument domain exception</td> </tr> <tr> <td>SING</td> <td>argument singularity</td> </tr> <tr> <td>OVERFLOW</td> <td>overflow range exception</td> </tr> <tr> <td>UNDERFLOW</td> <td>underflow range exception</td> </tr> <tr> <td>TLOSS</td> <td>total loss of significance</td> </tr> <tr> <td>PLOSS</td> <td>partial loss of significance</td> </tr> </table> <p>Note that both <code>TLOSS</code> and <code>PLOSS</code> reflect limitations of particular algorithms for trigonometric functions that suffer abrupt declines in accuracy at definite boundaries. Since the implementation does not suffer such abrupt declines, <code>PLOSS</code> is never signaled. <code>TLOSS</code> is signaled for Bessel functions <i>only</i> to satisfy SVID3 requirements.</p> <p>The <code>name</code> member points to a string containing the name of the function that incurred the exception. The <code>arg1</code> and <code>arg2</code> members are the arguments with which the function was invoked. <code>retval</code> is set to the default value that will be returned by the function unless the user's <code>matherr()</code> sets it to a different value.</p> | DOMAIN | argument domain exception | SING | argument singularity | OVERFLOW | overflow range exception | UNDERFLOW | underflow range exception | TLOSS | total loss of significance | PLOSS | partial loss of significance |
| DOMAIN | argument domain exception | | | | | | | | | | | | |
| SING | argument singularity | | | | | | | | | | | | |
| OVERFLOW | overflow range exception | | | | | | | | | | | | |
| UNDERFLOW | underflow range exception | | | | | | | | | | | | |
| TLOSS | total loss of significance | | | | | | | | | | | | |
| PLOSS | partial loss of significance | | | | | | | | | | | | |

**SVID3
STANDARD
CONFORMANCE**

If the user's **matherr()** function returns non-zero, no exception message will be printed, and `errno` will not be set.

When an application is built as a SVID3 conforming application (see **standards(5)**), if **matherr()** is not supplied by the user, the default `matherr` exception-handling mechanisms, summarized in the table below, will be invoked upon exception:

| | |
|-----------|--|
| DOMAIN | 0.0 is usually returned, <code>errno</code> is set to EDOM, and a message is usually printed on standard error. |
| SING | The largest finite single-precision number, HUGE of appropriate sign is returned, <code>errno</code> is set to EDOM, and a message is printed on standard error. |
| OVERFLOW | The largest finite single-precision number, HUGE of appropriate sign is usually returned, <code>errno</code> is set to ERANGE. |
| UNDERFLOW | 0.0 is returned, and <code>errno</code> is set to ERANGE. |
| TLOSS | 0.0 is returned, <code>errno</code> is set to ERANGE, and a message is printed on standard error. |

In general, `errno` is not a reliable error indicator in that it may be unexpectedly set by a function in a handler for an asynchronous signal.

| SVID3 ERROR HANDLING PROCEDURES (compile with <code>cc -Xt</code>) | | | | | |
|---|-------------------------|--------------------------|--------------------------|---------------------------|--------|
| Types of Errors | | | | | |
| <math.h> type | DOMAIN | SING | OVERFLOW | UNDERFLOW | TLOSS |
| <code>errno</code> | EDOM | EDOM | ERANGE | ERANGE | ERANGE |
| IEEE Exception | Invalid Operation | Division by Zero | Overflow | Underflow | – |
| <code>fp_exception_type</code> | <code>fp_invalid</code> | <code>fp_division</code> | <code>fp_overflow</code> | <code>fp_underflow</code> | – |
| ACOS, ASIN ($ x > 1$): | Md, 0.0 | – | – | – | – |
| ACOSH ($x < 1$), ATANH ($ x > 1$): | NaN | – | – | – | – |
| ATAN2 (0,0): | Md, 0.0 | – | – | – | – |

| | | | | | |
|-------------------------------------|--------------|--------------|-----------|------|---------|
| COSH, SINH: | - | - | ±HUGE | - | - |
| EXP: | - | - | +HUGE | 0.0 | - |
| FMOD (x,0): | x | - | - | - | - |
| HYPOT: | - | - | +HUGE | - | - |
| J0, J1, JN (x > X_TLOSS): | - | - | - | - | Mt, 0.0 |
| LGAMMA: usual cases | - | - | +HUGE | - | - |
| (x = 0 or -integer) | - | Ms, +HUGE | - | - | - |
| LOG, LOG10: (x < 0) | Md, -HUGE | - | - | - | - |
| (x = 0) | - | Ms, -HUGE | - | - | - |
| POW: usual cases | - | - | ±HUGE | ±0.0 | - |
| (x < 0) ** (y not an integer) | Md, 0.0 | - | - | - | - |
| 0 ** 0 | Md, 0.0 | - | - | - | - |
| 0 ** (y < 0) | Md, 0.0 | - | - | - | - |
| REMAINDER (x,0)NaN | - | - | - | - | - |
| SCALB: | - | - | ±HUGE_VAL | ±0.0 | - |
| SQRT (x < 0): | Md, 0.0 | - | - | - | - |
| Y0, Y1, YN: (x < 0) | Md, -HUGE | - | - | - | - |

| | | | | | |
|------------------|---|--------------|---|---|---------|
| (x = 0) | - | Md, -HUGE | - | - | - |
| (x > X_TLOSS) | - | - | - | - | Mt, 0.0 |

| ABBREVIATIONS | |
|---------------|--|
| Md | Message is printed (DOMAIN error). |
| Ms | Message is printed (SING error). |
| Mt | Message is printed (TLOSS error). |
| NaN | IEEE NaN result and invalid operation exception. |
| HUGE | Maximum finite single-precision floating-point number. |
| HUGE_VAL | IEEE ∞ result and division-by-zero exception. |
| X_TLOSS | The value X_TLOSS is defined in <values.h>. |

The interaction of IEEE arithmetic and **matherr()** is not defined when executing under IEEE rounding modes other than the default round to nearest: **matherr()** is not always called on overflow or underflow, and the **matherr()** may return results that differ from those in this table.

**X/OPEN
COMMON
APPLICATION
ENVIRONMENT
(CAE)
SPECIFICATIONS
CONFORMANCE**

The X/Open System Interfaces and Headers (XSH) Issue 3 and later revisions of that specification no longer sanctions the use of the `matherr()` interface. The following table summarizes the values returned in the exceptional cases. In general, XSH dictates that as long as one of the input argument(s) is a NaN, NaN shall be returned. In particular, `pow(NaN, 0) = NaN`.

| CAE SPECIFICATION ERROR HANDLING PROCEDURES (compile with <code>cc -xa</code>) | | | | | |
|---|--------|------|----------|-----------|--------|
| Types of Errors | | | | | |
| <math.h> type | DOMAIN | SING | OVERFLOW | UNDERFLOW | TLOSS |
| errno | EDOM | EDOM | ERANGE | ERANGE | ERANGE |
| ACOS, ASIN(x > 1): | 0.0 | - | - | - | - |
| ATAN2(0,0): | 0.0 | - | - | - | - |

| | | | | | |
|-------------------------------------|-------------|-----------|-------------|-------|-------|
| COSH, SINH: | - | - | {±HUGE_VAL} | - | - |
| EXP: | - | - | {+HUGE_VAL} | {0.0} | - |
| FMOD (x,0): | {NaN} | - | - | - | - |
| HYPOT: | - | - | {+HUGE_VAL} | - | - |
| J0, J1, JN (x > X_TLOSS): | - | - | - | - | {0.0} |
| LGAMMA: usual cases | - | - | {+HUGE_VAL} | - | - |
| (x = 0 or -integer) | - | +HUGE_VAL | - | - | - |
| LOG, LOG10: | | | | | |
| (x < 0) | -HUGE_VAL | - | - | - | - |
| (x = 0) | - | -HUGE_VAL | - | - | - |
| POW: | | | | | |
| usual cases | - | - | ±HUGE_VAL | ±0.0 | - |
| (x < 0) ** (y not an integer) | 0.0 | - | - | - | - |
| 0 ** 0 | {1.0} | - | - | - | - |
| 0 ** (y < 0) | {-HUGE_VAL} | - | - | - | - |
| SQRT (x < 0): | 0.0 | - | - | - | - |
| Y0, Y1, YN: | | | | | |
| (x < 0) | {-HUGE_VAL} | - | - | - | - |

| | | | | | |
|---------------|---|-------------|---|---|-----|
| (x = 0) | - | {-HUGE_VAL} | - | - | - |
| (x > X_TLOSS) | - | - | - | - | 0.0 |

| ABBREVIATIONS | |
|---------------|---|
| {...} | errno is not to be relied upon in all braced cases. |
| NaN | IEEE NaN result and invalid operation exception. |
| HUGE_VAL | IEEE ∞ result and division-by-zero exception. |
| X_TLOSS | The value X_TLOSS is defined in <values.h>. |

**ANSI/ISO-C
STANDARD
CONFORMANCE**

The ANSI/ISO-C standard covers a small subset of the CAE specification.

The following table summarizes the values returned in the exceptional cases.

| ANSI/ISO-C ERROR HANDLING PROCEDURES (compile with cc -xc) | | | | |
|--|-----------|-----------|-----------|-----------|
| Types of Errors | | | | |
| <math.h> type | DOMAIN | SING | OVERFLOW | UNDERFLOW |
| errno | EDOM | EDOM | ERANGE | ERANGE |
| ACOS, ASIN (x > 1): | 0.0 | - | - | - |
| ATAN2 (0,0): | 0.0 | - | - | - |
| EXP: | - | - | +HUGE_VAL | 0.0 |
| FMOD (x,0): | NaN | - | - | - |
| LOG, LOG10: | | | | |
| (x < 0) | -HUGE_VAL | - | - | - |
| (x = 0) | - | -HUGE_VAL | - | - |
| POW: | | | | |
| usual cases | - | - | ±HUGE_VAL | ±0.0 |
| (x < 0) ** (y not an integer) | 0.0 | - | - | - |

| | | | | |
|-----------------|-----------|---|---|---|
| $0^{**}(y < 0)$ | -HUGE_VAL | - | - | - |
| SQRT(x < 0): | 0.0 | - | - | - |

| ABBREVIATIONS | |
|---------------|--|
| NaN | IEEE NaN result and invalid operation exception. |
| HUGE_VAL | IEEE ∞ result and division-by-zero exception. |

EXAMPLES

EXAMPLE 1 Example of **matherr()** function.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int
matherr(struct exception *x) {
    switch (x->type) {
        case DOMAIN:
            /* change sqrt to return sqrt(-arg1), not NaN */
            if (!strcmp(x->name, "sqrt")) {
                x->retval = sqrt(-x->arg1);
                return (0); /* print message and set errno */
            } /* FALLTHRU */
        case SING:
            /* all other domain or sing exceptions, print message and */
            /* abort */
            fprintf(stderr, "domain exception in %s\n", x->name);
            abort( );
            break;
    }
    return (0); /* all other exceptions, execute default procedure */
}
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

attributes(5), **standards(5)**

| NAME | mblen – get number of bytes in a character | | | | | | |
|----------------------|---|----------------|-----------------|----------|-------------------------|-----|---------|
| SYNOPSIS | <pre>#include <stdlib.h> int mblen(const char *s, size_t n);</pre> | | | | | | |
| DESCRIPTION | <p>If <i>s</i> is not a null pointer, mblen() determines the number of bytes constituting the character pointed to by <i>s</i>. It is equivalent to:</p> <pre>mbtowc((wchar_t *)0, s, n);</pre> <p>A call with <i>s</i> as a null pointer causes this function to return 0. The behavior of this function is affected by the LC_CTYPE category of the current locale.</p> | | | | | | |
| RETURN VALUES | <p>If <i>s</i> is a null pointer, mblen() returns 0. If <i>s</i> is not a null pointer, mblen() returns 0 (if <i>s</i> points to the null byte), the number of bytes that constitute the character (if the next <i>n</i> or fewer bytes form a valid character), or -1 (if they do not form a valid character) and may set <code>errno</code> to indicate the error. In no case will the value returned be greater than <i>n</i> or the value of the <code>MB_CUR_MAX</code> macro.</p> | | | | | | |
| ERRORS | <p>The mblen() function may fail if:</p> <p>EILSEQ Invalid character sequence is detected.</p> | | | | | | |
| USAGE | <p>The mblen() function can be used safely in multithreaded applications, as long as setlocale(3C) is not being called to change the locale.</p> | | | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" data-bbox="485 1272 1385 1402"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe with exceptions</td> </tr> <tr> <td>CSI</td> <td>Enabled</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe with exceptions | CSI | Enabled |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| MT-Level | MT-Safe with exceptions | | | | | | |
| CSI | Enabled | | | | | | |
| SEE ALSO | <p>mbstowcs(3C), mbtowc(3C), setlocale(3C), wcstombs(3C), wctomb(3C), attributes(5)</p> | | | | | | |

| | |
|----------------------|---|
| NAME | mbrlen – get number of bytes in a character (restartable) |
| SYNOPSIS | <pre>#include <wchar.h> size_t mbrlen(const char *s, size_t n, mbstate_t *ps);</pre> |
| DESCRIPTION | <p>If <i>s</i> is not a null pointer, mbrlen() determines the number of bytes constituting the character pointed to by <i>s</i>. It is equivalent to:</p> <pre>mbstate_t internal; mbrtowc(NULL, s, n, ps != NULL ? ps : &internal);</pre> <p>If <i>ps</i> is a null pointer, the mbrlen() function uses its own internal <code>mbstate_t</code> object, which is initialized at program startup to the initial conversion state. Otherwise, the <code>mbstate_t</code> object pointed to by <i>ps</i> is used to completely describe the current conversion state of the associated character sequence. Solaris will behave as if no function defined in the Solaris Reference Manual calls mbrlen().</p> <p>The behavior of this function is affected by the <code>LC_CTYPE</code> category of the current locale. See environ(5).</p> |
| RETURN VALUES | <p>The mbrlen() function returns the first of the following that applies:</p> <ul style="list-style-type: none"> 0 If the next <i>n</i> or fewer bytes complete the character that corresponds to the null wide-character. positive If the next <i>n</i> or fewer bytes complete a valid character; the value returned is the number of bytes that complete the character. (size_t)-2 If the next <i>n</i> bytes contribute to an incomplete but potentially valid character, and all <i>n</i> bytes have been processed. When <i>n</i> has at least the value of the <code>MB_CUR_MAX</code> macro, this case can only occur if <i>s</i> points at a sequence of redundant shift sequences (for implementations with state-dependent encodings). (size_t)-1 If an encoding error occurs, in which case the next <i>n</i> or fewer bytes do not contribute to a complete and valid character. In this case, <code>EILSEQ</code> is stored in <code>errno</code> and the conversion state is undefined. |
| ERRORS | The mbrlen() function may fail if: |

EINVAL The *ps* argument points to an object that contains an invalid conversion state.

EILSEQ Invalid character sequence is detected.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | See NOTES below |

SEE ALSO

mbrtowc(3C), **mbsinit(3C)**, **setlocale(3C)**, **attributes(5)**, **environ(5)**

NOTES

If *ps* is not a null pointer, **mbrlen()** uses the `mbstate_t` object pointed to by *ps* and the function can be used safely in multithreaded applications, as long as **setlocale(3C)** is not being called to change the locale. If *ps* is a null pointer, **mbrlen()** uses its internal `mbstate_t` object and the function is Unsafe in multithreaded applications.

| | |
|----------------------|---|
| NAME | mbrtowc – convert a character to a wide-character code (restartable) |
| SYNOPSIS | <pre>#include <wchar.h> size_t mbrtowc(wchar_t *pwc, const char *s, size_t n, mbstate_t *ps);</pre> |
| DESCRIPTION | <p>If <i>s</i> is a null pointer, the mbrtowc() function is equivalent to the call:</p> <pre>mbrtowc(NULL, '', 1, ps)</pre> <p>In this case, the values of the arguments <i>pwc</i> and <i>n</i> are ignored.</p> <p>If <i>s</i> is not a null pointer, the mbrtowc() function inspects at most <i>n</i> bytes beginning at the byte pointed to by <i>s</i> to determine the number of bytes needed to complete the next character (including any shift sequences). If the function determines that the next character is completed, it determines the value of the corresponding wide-character and then, if <i>pwc</i> is not a null pointer, stores that value in the object pointed to by <i>pwc</i>. If the corresponding wide-character is the null wide-character, the resulting state described is the initial conversion state.</p> <p>If <i>ps</i> is a null pointer, the mbrtowc() function uses its own internal <code>mbstate_t</code> object, which is initialized at program startup to the initial conversion state. Otherwise, the <code>mbstate_t</code> object pointed to by <i>ps</i> is used to completely describe the current conversion state of the associated character sequence. Solaris will behave as if no function defined in the Solaris Reference Manual calls mbrtowc().</p> <p>The behavior of this function is affected by the <code>LC_CTYPE</code> category of the current locale. See environ(5).</p> |
| RETURN VALUES | <p>The mbrtowc() function returns the first of the following that applies:</p> <ul style="list-style-type: none"> 0 If the next <i>n</i> or fewer bytes complete the character that corresponds to the null wide-character (which is the value stored). positive If the next <i>n</i> or fewer bytes complete a valid character (which is the value stored); the value returned is the number of bytes that complete the character. (<code>size_t</code>)-2 If the next <i>n</i> bytes contribute to an incomplete but potentially valid character, and all <i>n</i> bytes have been processed (no value is stored). When <i>n</i> has at least the value of the <code>MB_CUR_MAX</code> macro, this case can only occur if <i>s</i> points at a sequence of redundant shift sequences (for implementations with state-dependent encodings). |

(`size_t`)-1 If an encoding error occurs, in which case the next *n* or fewer bytes do not contribute to a complete and valid character (no value is stored). In this case, `EILSEQ` is stored in `errno` and the conversion state is undefined.

ERRORS

The `mbrtowc()` function may fail if:

EINVAL The *ps* argument points to an object that contains an invalid conversion state.

EILSEQ Invalid character sequence is detected.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|------------------------------|
| MT-Level | See <code>NOTES</code> below |

SEE ALSO

`mbsinit(3C)`, `setlocale(3C)`, `attributes(5)`, `environ(5)`

NOTES

If *ps* is not a null pointer, `mbrtowc()` uses the `mbstate_t` object pointed to by *ps* and the function can be used safely in multithreaded applications, as long as `setlocale(3C)` is not being called to change the locale. If *ps* is a null pointer, `mbrtowc()` uses its internal `mbstate_t` object and the function is Unsafe in multithreaded applications.

| NAME | mbsinit – determine conversion object status | | | | |
|----------------------|---|----------------|-----------------|----------|-------------------------|
| SYNOPSIS | <pre>#include <wchar.h> int mbsinit(const mbstate_t *ps);</pre> | | | | |
| DESCRIPTION | If <code>ps</code> is not a null pointer, the mbsinit() function determines whether the object pointed to by <code>ps</code> describes an initial conversion state. | | | | |
| RETURN VALUES | <p>The mbsinit() function returns non-zero if <code>ps</code> is a null pointer, or if the pointed-to object describes an initial conversion state; otherwise, it returns 0.</p> <p>If an <code>mbstate_t</code> object is altered by any of the functions described as "restartable", and is then used with a different character sequence, or in the other conversion direction, or with a different <code>LC_CTYPE</code> category setting than on earlier function calls, the behavior is undefined. See environ(5).</p> | | | | |
| ERRORS | No errors are defined. | | | | |
| USAGE | <p>The <code>mbstate_t</code> object is used to describe the current conversion state from a particular character sequence to a wide-character sequence (or vice versa) under the rules of a particular setting of the <code>LC_CTYPE</code> category of the current locale.</p> <p>The initial conversion state corresponds, for a conversion in either direction, to the beginning of a new character sequence in the initial shift state. A zero-valued <code>mbstate_t</code> object is at least one way to describe an initial conversion state. A zero-valued <code>mbstate_t</code> object can be used to initiate conversion involving any character sequence, in any <code>LC_CTYPE</code> category setting.</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe with exceptions</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe with exceptions |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe with exceptions | | | | |
| SEE ALSO | mbrlen(3C) , mbrtowc(3C) , mbsrtowcs(3C) , setlocale(3C) , wcrtomb(3C) , wcsrtombs(3C) , attributes(5) , environ(5) | | | | |
| NOTES | The mbsinit() function can be used safely in multithreaded applications, as long as setlocale(3C) is not being called to change the locale. | | | | |

| | | | | | |
|----------------------|---|---------------|---|---------------|---|
| NAME | mbsrtowcs – convert a character string to a wide-character string (restartable) | | | | |
| SYNOPSIS | <pre>#include <wchar.h> size_t mbsrtowcs(wchar_t *dst, const char **src, size_t len, mbstate_t *ps);</pre> | | | | |
| DESCRIPTION | <p>The mbsrtowcs() function converts a sequence of characters, beginning in the conversion state described by the object pointed to by <i>ps</i>, from the array indirectly pointed to by <i>src</i> into a sequence of corresponding wide-characters. If <i>dst</i> is not a null pointer, the converted characters are stored into the array pointed to by <i>dst</i>. Conversion continues up to and including a terminating null character, which is also stored. Conversion stops early in either of the following cases:</p> <ul style="list-style-type: none"> ■ When a sequence of bytes is encountered that does not form a valid character. ■ When <i>len</i> codes have been stored into the array pointed to by <i>dst</i> (and <i>dst</i> is not a null pointer). <p>Each conversion takes place as if by a call to the mbrtowc() function.</p> <p>If <i>dst</i> is not a null pointer, the pointer object pointed to by <i>src</i> is assigned either a null pointer (if conversion stopped due to reaching a terminating null character) or the address just past the last character converted (if any). If conversion stopped due to reaching a terminating null character, and if <i>dst</i> is not a null pointer, the resulting state described is the initial conversion state.</p> <p>If <i>ps</i> is a null pointer, the mbsrtowcs() function uses its own internal <code>mbstate_t</code> object, which is initialized at program startup to the initial conversion state. Otherwise, the <code>mbstate_t</code> object pointed to by <i>ps</i> is used to completely describe the current conversion state of the associated character sequence. Solaris will behave as if no function defined in the Solaris Reference Manual calls mbsrtowcs().</p> <p>The behavior of this function is affected by the LC_CTYPE category of the current locale. See environ(5).</p> | | | | |
| RETURN VALUES | <p>If the input conversion encounters a sequence of bytes that do not form a valid character, an encoding error occurs. In this case, the mbsrtowcs() function stores the value of the macro <code>EILSEQ</code> in <code>errno</code> and returns <code>(size_t)-1</code>; the conversion state is undefined. Otherwise, it returns the number of characters successfully converted, not including the terminating null (if any).</p> | | | | |
| ERRORS | <p>The mbsrtowcs() function may fail if:</p> <table border="0" style="width: 100%;"> <tr> <td style="padding-right: 20px;">EINVAL</td> <td>The <i>ps</i> argument points to an object that contains an invalid conversion state.</td> </tr> <tr> <td>EILSEQ</td> <td>Invalid character sequence is detected.</td> </tr> </table> | EINVAL | The <i>ps</i> argument points to an object that contains an invalid conversion state. | EILSEQ | Invalid character sequence is detected. |
| EINVAL | The <i>ps</i> argument points to an object that contains an invalid conversion state. | | | | |
| EILSEQ | Invalid character sequence is detected. | | | | |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | See NOTES below |

SEE ALSO

mbrtowc(3C), **mbsinit(3C)**, **setlocale(3C)**, **attributes(5)**, **environ(5)**

NOTES

If *ps* is not a null pointer, **mbsrtowcs()** uses the `mbstate_t` object pointed to by *ps* and the function can be used safely in multithreaded applications, as long as **setlocale(3C)** is not being called to change the locale. If *ps* is a null pointer, **mbsrtowcs()** uses its internal `mbstate_t` object and the function is Unsafe in multithreaded applications.

| NAME | mbstowcs – convert a character string to a wide-character string | | | | | | |
|----------------------|--|----------------|-----------------|----------|---------|-----|---------|
| SYNOPSIS | <pre>#include <stdlib.h> size_t mbstowcs(wchar_t *pwcs, const char *s, size_t n);</pre> | | | | | | |
| DESCRIPTION | <p>The mbstowcs() function converts a sequence of characters from the array pointed to by <i>s</i> into a sequence of corresponding wide-character codes and stores not more than <i>n</i> wide-character codes into the array pointed to by <i>pwcs</i>. No characters that follow a null byte (which is converted into a wide-character code with value 0) will be examined or converted. Each character is converted as if by a call to mbtowc(3C).</p> <p>to cut to</p> <p>No more than <i>n</i> elements will be modified in the array pointed to by <i>pwcs</i>. If copying takes place between objects that overlap, the behavior is undefined.</p> <p>The behavior of this function is affected by the LC_CTYPE category of the current locale. If <i>pwcs</i> is a null pointer, mbstowcs() returns the length required to convert the entire array regardless of the value of <i>n</i>, but no values are stored.</p> | | | | | | |
| RETURN VALUES | <p>If an invalid character is encountered, mbstowcs() returns $(\text{size_t})-1$ and may set <code>errno</code> to indicate the error. Otherwise, mbstowcs() returns the number of the array elements modified (or required if <i>pwcs</i> is NULL), not including a terminating 0 code, if any. The array will not be zero-terminated if the value returned is <i>n</i>.</p> | | | | | | |
| ERRORS | <p>The mbstowcs() function may fail if:</p> <p>EILSEC Invalid byte sequence is detected.</p> | | | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> <tr> <td>CSI</td> <td>Enabled</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe | CSI | Enabled |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| MT-Level | MT-Safe | | | | | | |
| CSI | Enabled | | | | | | |
| SEE ALSO | <p>mblen(3C), mbtowc(3C), setlocale(3C), wcstombs(3C), wctomb(3C), attributes(5)</p> | | | | | | |

| NAME | mbtowc – convert a character to a wide-character code | | | | | | |
|----------------------|--|----------------|-----------------|----------|-------------------------|-----|---------|
| SYNOPSIS | <pre>#include <stdlib.h> int mbtowc(wchar_t *pwc, const char *s, size_t n);</pre> | | | | | | |
| DESCRIPTION | <p>If <i>s</i> is not a null pointer, mbtowc() determines the number of the bytes that constitute the character pointed to by <i>s</i>. It then determines the wide-character code for the value of type <code>wchar_t</code> that corresponds to that character. (The value of the wide-character code corresponding to the null byte is 0.) If the character is valid and <i>pwc</i> is not a null pointer, mbtowc() stores the wide-character code in the object pointed to by <i>pwc</i>.</p> <p>A call with <i>s</i> as a null pointer causes this function to return 0. The behavior of this function is affected by the LC_CTYPE category of the current locale. At most <i>n</i> bytes of the array pointed to by <i>s</i> will be examined.</p> | | | | | | |
| RETURN VALUES | <p>If <i>s</i> is a null pointer, mbtowc() returns 0. If <i>s</i> is not a null pointer, mbtowc() returns 0 (if <i>s</i> points to the null byte), the number of bytes that constitute the converted character (if the next <i>n</i> or fewer bytes form a valid character), or <code>-1</code> and may set <code>errno</code> to indicate the error (if they do not form a valid character).</p> <p>In no case will the value returned be greater than <i>n</i> or the value of the <code>MB_CUR_MAX</code> macro.</p> | | | | | | |
| ERRORS | <p>The mbtowc() function may fail if:</p> <p>EILSEQ Invalid character sequence is detected.</p> | | | | | | |
| USAGE | <p>The mbtowc() function can be used safely in multithreaded applications, as long as setlocale(3C) is not being called to change the locale.</p> | | | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe with exceptions</td> </tr> <tr> <td>CSI</td> <td>Enabled</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe with exceptions | CSI | Enabled |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| MT-Level | MT-Safe with exceptions | | | | | | |
| CSI | Enabled | | | | | | |
| SEE ALSO | <p>mblen(3C), mbstowcs(3C), setlocale(3C), wcstombs(3C), wctomb(3C), attributes(5)</p> | | | | | | |

| | |
|--------------------|---|
| NAME | mctl – memory management control |
| SYNOPSIS | <pre> /usr/ucb/cc[<i>flag ...</i>] <i>file ...</i> #include <sys/types.h> #include <sys/mman.h> int mctl(<i>addr, len, function, arg</i>); caddr_t <i>addr</i>; size_t <i>len</i>; int <i>function</i>; int <i>arg</i>; </pre> |
| DESCRIPTION | <p>mctl() applies a variety of control functions over pages identified by the mappings established for the address range [<i>addr</i>, <i>addr + len</i>). The function to be performed is identified by the argument <i>function</i>. Valid functions are defined in <code>mman.h</code> as follows:</p> <p>MC_LOCK Lock the pages in the range in memory. This function is used to support mlock(). See mlock(3C) for semantics and usage. <i>arg</i> is ignored.</p> <p>MC_LOCKAS Lock the pages in the address space in memory. This function is used to support mlockall(). See mlockall(3C) for semantics and usage. <i>addr</i> and <i>len</i> are ignored. <i>arg</i> is an integer built from the flags:</p> <p style="padding-left: 40px;">MCL_CURRENT Lock current mappings</p> <p style="padding-left: 40px;">MCL_FUTURE Lock future mappings</p> <p>MC_SYNC Synchronize the pages in the range with their backing storage. Optionally invalidate cache copies. This function is used to support msync(). See msync(3C) for semantics and usage. <i>arg</i> is used to represent the <i>flags</i> argument to msync(). It is constructed from an OR of the following values:</p> <p style="padding-left: 40px;">MS_SYNC Synchronized write</p> <p style="padding-left: 40px;">MS_ASYNC Return immediately</p> <p style="padding-left: 40px;">MS_INVALIDATE Invalidate mappings</p> <p>MS_ASYNC returns after all I/O operations are scheduled. MS_SYNC does not return until all I/O operations are complete. Specify exactly one of MS_ASYNC or MS_SYNC. MS_INVALIDATE invalidates all cached copies of data from</p> |

memory, requiring them to be re-obtained from the object's permanent storage location upon the next reference.

- MC_UNLOCK** Unlock the pages in the range. This function is used to support **munlock()**. *arg* is ignored.
- MC_UNLOCKAS** Remove address space memory lock, and locks on all current mappings. This function is used to support **munlockall()**. *addr* and *len* must have the value 0. *arg* is ignored.

RETURN VALUES

mctl() returns 0 on success, -1 on failure.

ERRORS

mctl() fails if:

- EAGAIN** Some or all of the memory identified by the operation could not be locked due to insufficient system resources.
- EBUSY** **MS_INVALIDATE** was specified and one or more of the pages is locked in memory.
- EINVAL** *addr* is not a multiple of the page size as returned by **getpagesize()**.
- EINVAL** *addr* and/or *len* do not have the value 0 when **MC_LOCKAS** or **MC_UNLOCKAS** are specified.
- EINVAL** *arg* is not valid for the function specified.
- EIO** An I/O error occurred while reading from or writing to the file system.
- ENOMEM** Addresses in the range [*addr*, *addr + len*) are invalid for the address space of a process, or specify one or more pages which are not mapped.
- EPERM** The process's effective user ID is not super-user and one of **MC_LOCK**, **MC_LOCKAS**, **MC_UNLOCK**, or **MC_UNLOCKAS** was specified.

SEE ALSO

mmap(2), **memcntl(2)**, **getpagesize(3C)**, **mlock(3C)**, **mlockall(3C)**, **msync(3C)**

NOTES

Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

| | |
|--------------------|--|
| NAME | media_findname – convert a supplied name into an absolute pathname that can be used to access removable media |
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lvolmgt [<i>library ...</i>] #include <volmgt.h> char *media_findname(char *<i>start</i>);</pre> |
| DESCRIPTION | <p>media_findname() converts the supplied <i>start</i> string into an absolute pathname that can then be used to access a particular piece of media.</p> <p>The <i>start</i> parameter can be one of the following types of specifications:</p> <p><i>/dev/...</i> An absolute pathname in <i>/dev</i>, such as <i>/dev/rdiskette0</i>, in which case a copy of that string is returned (see NOTES on this page).</p> <p><i>/vol/...</i> An absolute Volume Management pathname, such as <i>/vol/dev/aliases/floppy0</i> or <i>/vol/dsk/fred</i>. If this supplied pathname is not a symbolic link, then a copy of that pathname is returned. If the supplied pathname is a symbolic link then it is dereferenced and a copy of that dereferenced pathname is returned.</p> <p><i>volume_name</i> The Volume Management volume name for a particular volume, such as <i>fred</i> (see fdformat(1) for a description of how to label floppies). In this case a pathname in the Volume Management namespace is returned.</p> <p><i>volmgt_symname</i> The Volume Management symbolic name for a device, such as <i>floppy0</i> or <i>cdrom2</i> (see volfs(7FS) for more information on Volume Management symbolic names), in which case a pathname in the Volume Management namespace is returned.</p> <p><i>media_type</i> The Volume Management generic media type name. For example, <i>floppy</i> or <i>cdrom</i>. In this case media_findname() looks for the first piece of media that matches that media type, starting at 0 (zero) and continuing on until a match is found (or some fairly large maximum number is reached). In this case, if a match is found, a copy of the pathname to the volume found is returned.</p> |

RETURN VALUES

Upon successful completion **media_findname()** returns a pointer to the pathname found. In the case of an error a null pointer is returned.

ERRORS

For cases where the supplied *start* parameter is an absolute pathname, **media_findname()** can fail, returning a null string pointer, if an **lstat(2)** of that supplied pathname fails. Also, if the supplied absolute pathname is a symbolic link, **media_findname()** can fail if a **readlink(2)** of that symbolic link fails, or if a **stat(2)** of the pathname pointed to by that symbolic link fails, or if any of the following is true:

ENXIO The specified absolute pathname was not a character special device, and it was not a directory with a character special device in it.

EXAMPLES

EXAMPLE 1 Sample programs of the **media_findname()** function.

The following example attempts to find what the Volume Management pathname is to a piece of media called fred. Notice that a **volmgt_check()** is done first (see the NOTES section on this page).

```
(void) volmgt_check(NULL);
if ((nm = media_findname("fred")) != NULL) {
    (void) printf("media named \"fred\" is at \"%s\\\"\\n", nm);
} else {
    (void) printf("media named \"fred\" not found\\n");
}
```

This example looks for whatever volume is in the first floppy drive, letting **media_findname()** call **volmgt_check()** if and only if no floppy is currently known to be the first floppy drive.

```
if ((nm = media_findname("floppy0")) != NULL) {
    (void) printf("path to floppy0 is \"%s\\\"\\n", nm);
} else {
    (void) printf("nothing in floppy0\\n");
}
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Unsafe |

SEE ALSO

cc(1B), **fdformat(1)**, **vold(1M)**, **lstat(2)**, **readlink(2)**, **stat(2)**, **free(3C)**, **malloc(3C)**, **volmgt_check(3X)**, **volmgt_inuse(3X)**,

`volmgt_root(3X)`, `volmgt_running(3X)`, `volmgt_symname(3X)`,
`attributes(5)`, `volfs(7FS)`

NOTES

If `media_findname()` cannot find a match for the supplied name, it performs a `volmgt_check(3X)` and tries again, so it can be more efficient to perform `volmgt_check()` before calling `media_findname()`.

Upon success `media_findname()` returns a pointer to string which has been allocated; this should be freed when no longer in use (see `free(3C)`).

| NAME | media_getattr, media_setattr – get and set media attributes | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------|--|---|--|-------|-------------|----------|----|---------------|---------------------------------|-----------|----|-------------------------|---------------|---------|----|---|---------------------------------------|------------|----|-----------------|---|--------------|----|-----------------|--|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lvolmgt [<i>library</i> ...] #include <volmgt.h> char *media_getattr(char *vol_path, char *attr); int media_setattr(char *vol_path, char *attr, char *value);</pre> | | | | | | | | | | | | | | | | | | | | | | | | |
| DESCRIPTION | <p>media_setattr() and media_getattr() respectively set and get attribute-value pairs (called properties) on a per-volume basis.</p> <p>Volume Management supports system properties and user properties. System properties are ones that Volume Management predefines. Some of these system properties are writable, but only by the user that owns the volume being specified, and some system properties are read only:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Attribute</th> <th style="text-align: left;">Writable</th> <th style="text-align: left;">Value</th> <th style="text-align: left;">Description</th> </tr> </thead> <tbody> <tr> <td>s-access</td> <td>RO</td> <td>"seq", "rand"</td> <td>.na sequential or random access</td> </tr> <tr> <td>s-density</td> <td>RO</td> <td>"low", "medium", "high"</td> <td>media density</td> </tr> <tr> <td>s-parts</td> <td>RO</td> <td>.na comma separated list of slice numbers</td> <td>.na list of partitions on this volume</td> </tr> <tr> <td>s-location</td> <td>RO</td> <td><i>pathname</i></td> <td>.na Volume Management pathname to media</td> </tr> <tr> <td>s-mejectable</td> <td>RO</td> <td>"true", "false"</td> <td>.na whether or not media is manually ejectable</td> </tr> </tbody> </table> | Attribute | Writable | Value | Description | s-access | RO | "seq", "rand" | .na sequential or random access | s-density | RO | "low", "medium", "high" | media density | s-parts | RO | .na comma separated list of slice numbers | .na list of partitions on this volume | s-location | RO | <i>pathname</i> | .na Volume Management pathname to media | s-mejectable | RO | "true", "false" | .na whether or not media is manually ejectable |
| Attribute | Writable | Value | Description | | | | | | | | | | | | | | | | | | | | | | |
| s-access | RO | "seq", "rand" | .na sequential or random access | | | | | | | | | | | | | | | | | | | | | | |
| s-density | RO | "low", "medium", "high" | media density | | | | | | | | | | | | | | | | | | | | | | |
| s-parts | RO | .na comma separated list of slice numbers | .na list of partitions on this volume | | | | | | | | | | | | | | | | | | | | | | |
| s-location | RO | <i>pathname</i> | .na Volume Management pathname to media | | | | | | | | | | | | | | | | | | | | | | |
| s-mejectable | RO | "true", "false" | .na whether or not media is manually ejectable | | | | | | | | | | | | | | | | | | | | | | |

| | | | |
|-------------|-----|-----------------|---|
| s-rmoneject | R/W | "true", "false" | .na should media access points be removed from database upon ejection |
| s-enxio | R/W | "true", "false" | .na if set return ENXIO when media access attempted |

Properties can also be defined by the user. In this case the value can be any string the user wishes.

RETURN VALUES

Upon successful completion **media_getattr()** returns a pointer to the value corresponding to the specified attribute. A null pointer is returned if the specified volume doesn't exist, if the specified attribute for that volume doesn't exist, if the specified attribute is boolean and its value is false, or if **malloc(3C)** fails to allocate space for the return value.

media_setattr() returns 1 upon success, and 0 upon failure.

ERRORS

Both **media_getattr()** and **media_setattr()** can fail returning a null pointer if an **open(2)** of the specified *vol_path* fails, if an **fstat(2)** of that pathname fails, or if that pathname is not a block or character special device.

media_getattr() can also fail if the specified attribute was not found, and **media_setattr()** can also fail if the caller doesn't have permission to set the attribute, either because it's a system attribute, or because the caller doesn't own the specified volume.

Additionally, either routine can fail returning the following error values:

ENXIO The Volume Management daemon, *vold*, is not running
EINTR The routine was interrupted by the user before finishing

EXAMPLES

EXAMPLE 1 Sample programs of **media_getattr()**.

The following example checks to see if the volume called *fred* that Volume Management is managing can be ejected via software, or if it can only be manually ejected:

```
if (media_getattr("/vol/rdisk/fred", "s-mejectable") != NULL) {
    (void) printf("\\"fred\\" must be manually ejected\
");
} else {
    (void) printf("software can eject \\"fred\\"");
};
```

```
}

```

This example shows setting the *s-enxio* property for the floppy volume currently in the first floppy drive:

```
int    res;
if ((res = media_setattr("/vol/dev/aliases/floppy0", "s-enxio",
    "true")) == 0) {
    (void) printf("can't set s-enxio flag for floppy0\
");
}
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

cc(1B), **vold(1M)**, **lstat(2)**, **open(2)**, **readlink(2)**, **stat(2)**, **free(3C)**, **malloc(3C)**, **media_findname(3X)**, **volmgt_check(3X)**, **volmgt_inuse(3X)**, **volmgt_root(3X)**, **volmgt_running(3X)**, **volmgt_syname(3X)**, **attributes(5)**

NOTES

Upon success **media_getattr()** returns a pointer to a string which has been allocated, and should be freed when no longer in use (see **free(3C)**).

NAME | media_getid – return the id of a piece of media

SYNOPSIS | `cc [flag ...] file ...-lvolgmt [library ...]`

| `#include <volmgt.h>`

| `ulonglong_t media_getid(char *vol_path);`

DESCRIPTION | **media_getid()** returns the *id* of a piece of media. Volume Management must be running. See **volmgt_running(3X)**.

PARAMETERS | **vol_path** Path to the block or character special device.

RETURN VALUES | **media_getid()** returns the *id* of the volume. This value is unique for each volume. If **media_getid()** returns 0, the *path* provided is not valid, for example, it is a block or char device.

EXAMPLES | **EXAMPLE 1** Using **media_getid()**

The following example first checks if Volume Management is running, then checks the volume management name space for *path*, and then returns the *id* for the piece of media.

```
char *path;
...
if (volmgt_running()) {
    if (volmgt_ownspath(path)) {
        (void) printf("id of %s is %lld\n",
            path, media_getid(path));
    }
}
```

If a program using **media_getid()** does not check whether or not Volume Management is running, then any NULL return value will be ambiguous, as it could mean that either Volume Management does not have *path* in its name space, or Volume Management is not running.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|------------------|-----------------|
| MT Level | Safe |
| Commitment Level | Public |

media_getid(3X)

Miscellaneous Library Functions

SEE ALSO | `volmgt_ownspath(3X)`, `volmgt_running(3X)`, `attributes(5)`

| | |
|--------------------|---|
| NAME | memory, memccpy, memchr, memcmp, memcpy, memmove, memset – memory operations |
| SYNOPSIS | <pre>#include <string.h> void *memccpy(void *s1, const void *s2, int c, size_t n); void *memchr(const void *s, int c, size_t n); int memcmp(const void *s1, const void *s2, size_t n); void *memcpy(void *s1, const void *s2, size_t n); void *memmove(void *s1, const void *s2, size_t n); void *memset(void *s, int c, size_t n);</pre> |
| DESCRIPTION | <p>These functions operate as efficiently as possible on memory areas (arrays of bytes bounded by a count, not terminated by a null character). They do not check for the overflow of any receiving memory area.</p> <p>The memccpy() function copies bytes from memory area <i>s2</i> into <i>s1</i>, stopping after the first occurrence of <i>c</i> (converted to an <code>unsigned char</code>) has been copied, or after <i>n</i> bytes have been copied, whichever comes first. It returns a pointer to the byte after the copy of <i>c</i> in <i>s1</i>, or a null pointer if <i>c</i> was not found in the first <i>n</i> bytes of <i>s2</i>.</p> <p>The memchr() function returns a pointer to the first occurrence of <i>c</i> (converted to an <code>unsigned char</code>) in the first <i>n</i> bytes (each interpreted as an <code>unsigned char</code>) of memory area <i>s</i>, or a null pointer if <i>c</i> does not occur.</p> <p>The memcmp() function compares its arguments, looking at the first <i>n</i> bytes (each interpreted as an <code>unsigned char</code>), and returns an integer less than, equal to, or greater than 0, according as <i>s1</i> is lexicographically less than, equal to, or greater than <i>s2</i> when taken to be unsigned characters.</p> <p>The memcpy() function copies <i>n</i> bytes from memory area <i>s2</i> to <i>s1</i>. It returns <i>s1</i>.</p> <p>The memmove() function copies <i>n</i> bytes from memory areas <i>s2</i> to <i>s1</i>. Copying between objects that overlap will take place correctly. It returns <i>s1</i>.</p> <p>The memset() function sets the first <i>n</i> bytes in memory area <i>s</i> to the value of <i>c</i> (converted to an <code>unsigned char</code>). It returns <i>s</i>.</p> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO `string(3C)`, `attributes(5)`

| | |
|--------------------|--|
| NAME | menu_attributes, set_menu_fore, menu_fore, set_menu_back, menu_back, set_menu_grey, menu_grey, set_menu_pad, menu_pad – control menus display attributes |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lmenu -lcurses [<i>library</i> ..] #include <menu.h> int set_menu_fore(MENU * menu, chtype attr); chtype menu_fore(MENU * menu); int set_menu_back(MENU * menu, chtype attr); chtype menu_back(MENU * menu); int set_menu_grey(MENU* menu, chtype attr); chtype menu_grey(MENU * menu); int set_menu_pad(MENU * menu, int pad); int menu_pad(MENU * menu);</pre> |
| DESCRIPTION | <p>set_menu_fore() sets the foreground attribute of <i>menu</i> — the display attribute for the current item (if selectable) on single-valued menus and for selected items on multi-valued menus. This display attribute is a <code>curses</code> library visual attribute. menu_fore() returns the foreground attribute of <i>menu</i> .</p> <p>set_menu_back() sets the background attribute of <i>menu</i> — the display attribute for unselected, yet selectable, items. This display attribute is a <code>curses</code> library visual attribute.</p> <p>set_menu_grey() sets the grey attribute of <i>menu</i> — the display attribute for nonselectable items in multi-valued menus. This display attribute is a <code>curses</code> library visual attribute. menu_grey() returns the grey attribute of <i>menu</i> .</p> <p>The pad character is the character that fills the space between the name and description of an item. set_menu_pad() sets the pad character for <i>menu</i> to <i>pad</i> . menu_pad() returns the pad character of <i>menu</i> .</p> |

RETURN VALUES

These routines return one of the following:

E_OK\011\011The routine returned successfully.
E_SYSTEM_ERROR\011\011System error.
E_BAD_ARGUMENT\011\011An incorrect argument was passed to the routine.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

courses(3X), **menus(3X)**, **attributes(5)**

NOTES

The header `<menu.h>` automatically includes the headers `<eti.h>` and `<courses.h>`.

NAME menu_cursor, pos_menu_cursor – correctly position a menu cursor

SYNOPSIS

```
cc
[
  flag
  ... ]
file
...
-lmenu

-lcurses
[
  library
  .. ]
#include <menu.h>

int pos_menu_cursor(MENU * menu);
```

DESCRIPTION **pos_menu_cursor()** moves the cursor in the window of *menu* to the correct position to resume menu processing. This is needed after the application calls a curses library I/O routine.

RETURN VALUES This routine returns one of the following:

```
E_OK\011\011The
routine returned successfully.
E_SYSTEM_ERROR\011\011System error.
E_BAD_ARGUMENT\011\011An incorrect argument was passed
\011\011to the routine.
E_NOT_POSTED\011\011The menu has not been posted.
```

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO **curses(3X)** , **menus(3X)** , **panel_update(3X)** , **panels(3X)** , **attributes(5)**

NOTES The header <menu.h> automatically includes the headers <eti.h> and <curses.h> .

| | |
|----------------------|--|
| NAME | menu_driver - command processor for the menus subsystem |
| SYNOPSIS | <pre>cc [flag ...] file ... -lmenu -lcurses [library ..] #include <menu.h> int menu_driver(MENU *menu, int c);</pre> |
| DESCRIPTION | <p>menu_driver() is the workhorse of the <code>menu</code> subsystem. It checks to determine whether the character <code>c</code> is a menu request or data. If <code>c</code> is a request, the menu driver executes the request and reports the result. If <code>c</code> is data (a printable ASCII character), it enters the data into the pattern buffer and tries to find a matching item. If no match is found, the menu driver deletes the character from the pattern buffer and returns <code>E_NO_MATCH</code>. If the character is not recognized, the menu driver assumes it is an application-defined command and returns <code>E_UNKNOWN_COMMAND</code>.</p> <p>Menu driver requests:</p> <pre>REQ_LEFT_ITEM Move left to an item. REQ_RIGHT_ITEM Move right to an item. REQ_UP_ITEM Move up to an item. REQ_DOWN_ITEM Move down to an item. REQ_SCR_ULINE Scroll up a line. REQ_SCR_DLINE Scroll down a line. REQ_SCR_DPAGE Scroll up a page. REQ_SCR_UPAGE Scroll down a page. REQ_FIRST_ITEM Move to the first item. REQ_LAST_ITEM Move to the last item. REQ_NEXT_ITEM Move to the next item. REQ_PREV_ITEM Move to the previous item. REQ_TOGGLE_ITEM Select/de-select an item. REQ_CLEAR_PATTERN Clear the menu pattern buffer. REQ_BACK_PATTERN Delete the previous character from pattern buffer. REQ_NEXT_MATCH Move the next matching item. REQ_PREV_MATCH Move to the previous matching item.</pre> |
| RETURN VALUES | <p>menu_driver() returns one of the following:</p> <pre>E_OK The routine returned successfully. E_SYSTEM_ERROR System error. E_BAD_ARGUMENT An incorrect argument was passed to the routine. E_BAD_STATE The routine was called from an initialization or termination function. E_NOT_POSTED The menu has not been posted. E_UNKNOWN_COMMAND An unknown request was passed to the menu driver. E_NO_MATCH The character failed to match. E_NOT_SELECTABLE The item cannot be selected.</pre> |

`E_REQUEST_DENIED` The menu driver could not process the request.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`curses(3X)`, `menus(3X)`, `attributes(5)`

NOTES

Application defined commands should be defined relative to (greater than) `MAX_COMMAND`, the maximum value of a request listed above.

The header `<menu.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

| NAME | menu_format, set_menu_format – set and get maximum numbers of rows and columns in menus | | | | |
|----------------------|---|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lmenu -lcurses [<i>library</i> ..] #include <menu.h> int set_menu_format(MENU * menu, int rows, int cols); void menu_format(MENU * menu, int * rows, int * cols);</pre> | | | | |
| DESCRIPTION | <p>set_menu_format() sets the maximum number of rows and columns of items that may be displayed at one time on a menu. If the menu contains more items than can be displayed at once, the menu will be scrollable.</p> <p>menu_format() returns the maximum number of rows and columns that may be displayed at one time on <i>menu</i>. <i>rows</i> and <i>cols</i> are pointers to the variables used to return these values.</p> | | | | |
| RETURN VALUES | <p>set_menu_format() returns one of the following:</p> <pre>E_OK\011\011The routine returned successfully. E_SYSTEM_ERROR\011\011System error. E_BAD_ARGUMENT\011\011An incorrect argument was passed \011\011to the routine. E_POSTED\011\011The menu is already posted.</pre> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | curses(3X), menus(3X), attributes(5) | | | | |

NOTES | The header `<menu.h>` automatically includes the headers `<eti.h>` and `<curses.h>` .

| | |
|--------------------|--|
| NAME | menu_hook, set_item_init, item_init, set_item_term, item_term, set_menu_init, menu_init, set_menu_term, menu_term – assign application-specific routines for automatic invocation by menus |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lmenu -lcurses [<i>library</i> ..] #include <menu.h> int set_item_init(MENU * menu, void (*func)(MENU *)); void (*item_init)(MENU * menu); int set_item_term(MENU * menu, void (*func)(MENU *)); void (*item_term)(MENU * menu); int set_menu_init(MENU * menu, void (*func)(MENU *));void (*menu_init)(MENU * menu); int set_menu_term(MENU * menu, void (*func)(MENU *));void (*menu_term)(MENU * menu);</pre> |
| DESCRIPTION | <p>set_item_init() assigns the application-defined function to be called when the <i>menu</i> is posted and just after the current item changes. item_init() returns a pointer to the item initialization routine, if any, called when the <i>menu</i> is posted and just after the current item changes.</p> <p>set_item_term() assigns an application-defined function to be called when the <i>menu</i> is unposted and just before the current item changes. item_term() returns a pointer to the termination function, if any, called when the <i>menu</i> is unposted and just before the current item changes.</p> <p>set_menu_init() assigns an application-defined function to be called when the <i>menu</i> is posted and just after the top row changes on a posted menu. menu_init() returns a pointer to the menu initialization routine, if any, called when the <i>menu</i> is posted and just after the top row changes on a posted menu.</p> <p>set_menu_term() assigns an application-defined function to be called when the <i>menu</i> is unposted and just before the top row changes on a posted menu.</p> |

menu_term() returns a pointer to the menu termination routine, if any, called when the *menu* is unposted and just before the top row changes on a posted menu.

RETURN VALUES

Routines that return pointers always return NULL on error. Routines that return an integer return one of the following:

E_OK\011\011The routine returned successfully.
E_SYSTEM_ERROR\011\011System error.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

curses(3X), **menus(3X)**, **attributes(5)**

NOTES

The header `<menu.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

| | |
|----------------------|---|
| NAME | menu_item_current, set_current_item, current_item, set_top_row, top_row, item_index – set and get current menus items |
| SYNOPSIS | <pre>cc [flag ...] file ... -lmenu -lcurses [library ..] #include <menu.h> int set_current_item(MENU * menu, ITEM * item); ITEM * current_item(MENU * menu); int set_top_row(MENU * menu, int row); int top_row(MENU * menu); int item_index(ITEM * item);</pre> |
| DESCRIPTION | <p>The current item of a menu is the item where the cursor is currently positioned. set_current_item() sets the current item of <i>menu</i> to <i>item</i> . current_item() returns a pointer to the the current item in <i>menu</i> .</p> <p>set_top_row() sets the top row of <i>menu</i> to <i>row</i> . The left-most item on the new top row becomes the current item. top_row() returns the number of the menu row currently displayed at the top of <i>menu</i> .</p> <p>item_index() returns the index to the <i>item</i> in the item pointer array. The value of this index ranges from 0 through $N - 1$, where N is the total number of items connected to the menu.</p> |
| RETURN VALUES | <p>current_item() returns NULL on error.</p> <p>top_row() and index_item() return -1 on error.</p> <p>set_current_item() and set_top_row() return one of the following:</p> <pre>E_OK\011\011The routine returned successfully. E_SYSTEM_ERROR\011\011System error. E_BAD_ARGUMENT\011\011An incorrect argument was passed \011\011to the routine.</pre> |

E_BAD_STATE\011\011The routine was called from an
\011\011initialization or termination function.
E_NOT_CONNECTED\011\011No items are connected to the menu.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

curses(3X) , **menus(3X)** , **attributes(5)**

NOTES

The header <menu.h> automatically includes the headers <eti.h> and <curses.h> .

NAME menu_item_name, item_name, item_description – get menus item name and description

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lmenu

-lcurses
[
library
.. ]
#include <menu.h>

char * item_name(ITEM * item);
char * item_description(ITEM * item);
```

DESCRIPTION **item_name()** returns a pointer to the name of *item* .
item_description() returns a pointer to the description of *item* .

RETURN VALUES These routines return NULL on error.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO **curses(3X)** , **menus(3X)** , **menu_new(3X)** , **attributes(5)**

NOTES The header <menu.h> automatically includes the headers <eti.h> and <curses.h> .

| NAME | menu_item_new, new_item, free_item – create and destroy menus items | | | | |
|----------------------|---|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lmenu -lcurses [<i>library</i> ..] #include <menu.h> ITEM * new_item(char * name, char * desc); int free_item(ITEM * item);</pre> | | | | |
| DESCRIPTION | <p>new_item() creates a new item from <i>name</i> and <i>description</i> , and returns a pointer to the new item.</p> <p>free_item() frees the storage allocated for <i>item</i> . Once an item is freed, the user can no longer connect it to a menu.</p> | | | | |
| RETURN VALUES | <p>new_item() returns NULL on error.</p> <p>free_item() returns one of the following:</p> <p>E_OK\011\011The routine returned successfully. E_SYSTEM_ERROR\011\011System error. E_BAD_ARGUMENT\011\011An incorrect argument was passed to \011\011the routine. E_CONNECTED\011\011One or more items are already connected \011\011to another menu.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | curses(3X) , menus(3X) , attributes(5) | | | | |

NOTES

The header `<menu.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

| | |
|----------------------|--|
| NAME | menu_item_opts, set_item_opts, item_opts_on, item_opts_off, item_opts – menus item option routines |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lmenu -lcurses [<i>library</i> ..] #include <menu.h> int set_item_opts(ITEM * item, OPTIONS opts); int item_opts_on(ITEM * item, OPTIONS opts); int item_opts_off(ITEM * item, OPTIONS opts); OPTIONS item_opts(ITEM * item);</pre> |
| DESCRIPTION | <p>set_item_opts() turns on the named options for <i>item</i> and turns off all other options. Options are boolean values that can be OR-ed together.</p> <p>item_opts_on() turns on the named options for <i>item</i> ; no other option is changed.</p> <p>item_opts_off() turns off the named options for <i>item</i> ; no other option is changed.</p> <p>item_opts() returns the current options of <i>item</i> .</p> <p>Item Options:</p> <p>O_SELECTABLE The item can be selected during menu processing.</p> |
| RETURN VALUES | <p>Except for item_opts() , these routines return one of the following:</p> <p>E_OK\011\011The routine returned successfully.</p> <p>E_SYSTEM_ERROR\011\011System error.</p> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `curses(3X)`, `menus(3X)`, `attributes(5)`

NOTES The header `<menu.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

| | |
|----------------------|---|
| NAME | menu_items, set_menu_items, item_count – connect and disconnect items to and from menus |
| SYNOPSIS | <pre>cc [flag ...] file ... -lmenu -lcurses [library ..] #include <menu.h> int set_menu_items(MENU * menu, ITEM ** items); ITEM ** menu_items(MENU * menu); int item_count(MENU * menu);</pre> |
| DESCRIPTION | <p>set_menu_items() changes the item pointer array connected to <i>menu</i> to the item pointer array <i>items</i> .</p> <p>menu_items() returns a pointer to the item pointer array connected to <i>menu</i> .</p> <p>item_count() returns the number of items in <i>menu</i> .</p> |
| RETURN VALUES | <p>menu_items() returns NULL on error.</p> <p>item_count() returns -1 on error.</p> <p>set_menu_items() returns one of the following:</p> <pre>E_OK\011\011The routine returned successfully. E_SYSTEM_ERROR\011\011System error. E_BAD_ARGUMENT\011\011An incorrect argument was passed to \011\011the routine. E_POSTED\011\011The menu is already posted. E_CONNECTED\011\011One or more items are already connected \011\011to another menu.</pre> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `curses(3X)`, `menus(3X)`, `attributes(5)`

NOTES The header `<menu.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

| NAME | menu_item_userptr, set_item_userptr, item_userptr – associate application data with menu items | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lmenu -lcurses [<i>library</i> ..] #include <menu.h> int set_item_userptr(ITEM * item, char * userptr); char * item_userptr(ITEM * item);</pre> | | | | |
| DESCRIPTION | Every item has an associated user pointer that can be used to store relevant information. set_item_userptr() sets the user pointer of <i>item</i> . item_userptr() returns the user pointer of <i>item</i> . | | | | |
| RETURN VALUES | <p>item_userptr() returns NULL on error. set_item_userptr() returns one of the following:</p> <pre>E_OK\011\011The routine returned successfully. E_SYSTEM_ERROR\011\011System error.</pre> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | curses(3X) , menus(3X) , attributes(5) | | | | |
| NOTES | The header <menu.h> automatically includes the headers <eti.h> and <curses.h>. | | | | |

NAME menu_item_value, set_item_value, item_value – set and get menus item values

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lmenu

-lcurses
[
library
.. ]
#include <menu.h>

int set_item_value(ITEM * item, int bool);

int item_value(ITEM * item);
```

DESCRIPTION

Unlike single-valued menus, multi-valued menus enable the end-user to select one or more items from a menu. **set_item_value()** sets the selected value of the *item* — TRUE (selected) or FALSE (not selected). **set_item_value()** may be used only with multi-valued menus. To make a menu multi-valued, use **set_menu_opts** or **menu_opts_off()** to turn off the option `O_ONEVALUE` . (See **menu_opts(3X)**).

item_value() returns the select value of *item* , either TRUE (selected) or FALSE (unselected).

RETURN VALUES

set_item_value() returns one of the following:

```
E_OK\011\011The routine
returned successfully.
E_SYSTEM_ERROR\011\011System error.
E_REQUEST_DENIED\011\011The menu driver could not process
\011\011the request.
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

curses(3X) , **menus(3X)** , **menu_opts(3X)** , **attributes(5)**

NOTES | The header `<menu.h>` automatically includes the headers `<eti.h>` and `<curses.h>` .

NAME menu_item_visible, item_visible – tell if menus item is visible

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lmenu

-lcurses
[
library
.. ]
#include <menu.h>

int item_visible(ITEM * item);
```

DESCRIPTION A menu item is visible if it currently appears in the subwindow of a posted menu. **item_visible()** returns TRUE if *item* is visible, otherwise it returns FALSE .

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO **curses(3X)** , **menus(3X)** , **menu_new(3X)** , **attributes(5)**

NOTES The header <menu.h> automatically includes the headers <eti.h> and <curses.h> .

| NAME | menu_mark, set_menu_mark – menus mark string routines | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lmenu -lcurses [library ..] #include <menu.h> int set_menu_mark(MENU * menu, char * mark); char * menu_mark(MENU * menu);</pre> | | | | |
| DESCRIPTION | menus displays mark strings to distinguish selected items in a menu (or the current item in a single-valued menu). set_menu_mark() sets the mark string of <i>menu</i> to <i>mark</i> . menu_mark() returns a pointer to the mark string of <i>menu</i> . | | | | |
| RETURN VALUES | <p>menu_mark() returns NULL on error. set_menu_mark() returns one of the following:</p> <pre>E_OK\011\011The routine returned successfully. E_SYSTEM_ERROR\011\011System error. E_BAD_ARGUMENT\011\011An incorrect argument was passed to \011\011the routine.</pre> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | curses(3X) , menus(3X) , attributes(5) | | | | |
| NOTES | The header <menu.h> automatically includes the headers <eti.h> and <curses.h> . | | | | |

| NAME | menu_new, new_menu, free_menu – create and destroy menus | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lmenu -lcurses [<i>library</i> ..] #include <menu.h> MENU * new_menu(ITEM ** <i>items</i>); int free_menu(MENU * <i>menu</i>);</pre> | | | | |
| DESCRIPTION | <p>new_menu() creates a new menu connected to the item pointer array <i>items</i> and returns a pointer to the new menu.</p> <p>free_menu() disconnects <i>menu</i> from its associated item pointer array and frees the storage allocated for the menu.</p> | | | | |
| RETURN VALUES | <p>new_menu() returns NULL on error.</p> <p>free_menu() returns one of the following:</p> <p>E_OK\011\011The routine returned successfully. E_SYSTEM_ERROR\011\011System error. E_BAD_ARGUMENT\011\011An incorrect argument was passed to \011\011the routine. E_POSTED\011\011The menu is already posted.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | curses(3X) , menus(3X) , attributes(5) | | | | |
| NOTES | The header <menu.h> automatically includes the headers <eti.h> and <curses.h> . | | | | |

| | | | | | | | | | | | |
|---------------------|--|------------|--|------------|---------------------------------------|------------|--------------------------------------|--------------|--|-------------|--|
| NAME | menu_opts, set_menu_opts, menu_opts_on, menu_opts_off – menus option routines | | | | | | | | | | |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lmenu -lcurses [<i>library</i> ..] #include <menu.h> OPTIONS menu_opts(MENU * menu); int set_menu_opts(MENU * menu, OPTIONS opts); int menu_opts_on(MENU * menu, OPTIONS opts); int menu_opts_off(MENU * menu, OPTIONS opts);</pre> | | | | | | | | | | |
| DESCRIPTION | | | | | | | | | | | |
| Menu Options | <p>set_menu_opts() turns on the named options for <i>menu</i> and turns off all other options. Options are boolean values that can be OR-ed together.</p> <p>menu_opts_on() turns on the named options for <i>menu</i> ; no other option is changed.</p> <p>menu_opts_off() turns off the named options for <i>menu</i> ; no other option is changed.</p> <p>menu_opts() returns the current options of <i>menu</i> .</p> <p>The following values can be OR'd together to create <i>opts</i> .</p> <table border="0"> <tr> <td style="padding-right: 20px;">O_ONEVALUE</td> <td>Only one item can be selected from the menu.</td> </tr> <tr> <td style="padding-right: 20px;">O_SHOWDESC</td> <td>Display the description of the items.</td> </tr> <tr> <td style="padding-right: 20px;">O_ROWMAJOR</td> <td>Display the menu in row major order.</td> </tr> <tr> <td style="padding-right: 20px;">O_IGNORECASE</td> <td>Ignore the case when pattern matching.</td> </tr> <tr> <td style="padding-right: 20px;">O_SHOWMATCH</td> <td>Place the cursor within the item name when pattern matching.</td> </tr> </table> | O_ONEVALUE | Only one item can be selected from the menu. | O_SHOWDESC | Display the description of the items. | O_ROWMAJOR | Display the menu in row major order. | O_IGNORECASE | Ignore the case when pattern matching. | O_SHOWMATCH | Place the cursor within the item name when pattern matching. |
| O_ONEVALUE | Only one item can be selected from the menu. | | | | | | | | | | |
| O_SHOWDESC | Display the description of the items. | | | | | | | | | | |
| O_ROWMAJOR | Display the menu in row major order. | | | | | | | | | | |
| O_IGNORECASE | Ignore the case when pattern matching. | | | | | | | | | | |
| O_SHOWMATCH | Place the cursor within the item name when pattern matching. | | | | | | | | | | |

RETURN VALUES

O_NONCYCLIC Make certain menu driver requests non-cyclic.

Except for **menu_opts()** , these routines return one of the following:

E_OK

The routine returned successfully.

E_SYSTEM_ERROR\011\011System error.

E_POSTED The menu is already posted.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

curses(3X) , **menus(3X)** , **attributes(5)**

NOTES

The header `<menu.h>` automatically includes the headers `<eti.h>` and `<curses.h>` .

| NAME | menu_pattern, set_menu_pattern – set and get menus pattern match buffer | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lmenu -lcurses [<i>library</i> ..] #include <menu.h> char *menu_pattern(MENU * menu); int set_menu_pattern(MENU * menu, char * pat);</pre> | | | | |
| DESCRIPTION | <p>Every menu has a pattern buffer to match entered data with menu items. set_menu_pattern() sets the pattern buffer to <i>pat</i> and tries to find the first item that matches the pattern. If it does, the matching item becomes the current item. If not, the current item does not change. menu_pattern() returns the string in the pattern buffer of <i>menu</i> .</p> | | | | |
| RETURN VALUES | <p>menu_pattern() returns NULL on error. set_menu_pattern() returns one of the following:</p> <pre>E_OK\011\011The routine returned successfully. E_SYSTEM_ERROR\011\011System error. E_BAD_ARGUMENT\011\011An incorrect argument was passed to \011\011the routine. E_NO_MATCH\011\011The character failed to match.</pre> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | curses(3X) , menus(3X) , attributes(5) | | | | |
| NOTES | The header <menu.h> automatically includes the headers <eti.h> and <curses.h> . | | | | |

| NAME | menu_post, post_menu, unpost_menu – write or erase menus from associated subwindows | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lmenu -lcurses [library ..] #include <menu.h> int post_menu(MENU * menu); int unpost_menu(MENU * menu);</pre> | | | | |
| DESCRIPTION | <p>post_menu() writes <i>menu</i> to the subwindow. The application programmer must use <i>curses</i> library routines to display the menu on the physical screen or call update_panels() if the <i>panels</i> library is being used.</p> <p>unpost_menu() erases <i>menu</i> from its associated subwindow.</p> | | | | |
| RETURN VALUES | <p>These routines return one of the following:</p> <p>E_OK The routine returned successfully.</p> <p>E_SYSTEM_ERROR System error.</p> <p>E_BAD_ARGUMENT An incorrect argument was passed to the routine.</p> <p>E_POSTED The menu is already posted.</p> <p>E_BAD_STATE The routine was called from an initialization or termination function.</p> <p>E_NO_ROOM The menu does not fit within its subwindow.</p> <p>E_NOT_POSTED The menu has not been posted.</p> <p>E_NOT_CONNECTED No items are connected to the menu.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |

SEE ALSO | `curses(3X)` , `menus(3X)` , `panels(3X)` , `attributes(5)`

NOTES | The header `<menu.h>` automatically includes the headers `<eti.h>` and `<curses.h>` .

NAME menus - character based menus package

SYNOPSIS #include <menu.h>

DESCRIPTION The menu library is built using the `curses` library, and any program using menus routines must call one of the `curses` initialization routines, such as `initscr`. A program using these routines must be compiled with `-lmenu` and `-lcurses` on the `cc` command line.

The menu package gives the applications programmer a terminal-independent method of creating and customizing menus for user interaction. The menu package includes: item routines, which are used to create and customize menu items; and menu routines, which are used to create and customize menus, assign pre- and post-processing routines, and display and interact with menus.

Current Default Values for Item Attributes The menu package establishes initial current default values for item attributes. During item initialization, each item attribute is assigned the current default value for that attribute. An application can change or retrieve a current default attribute value by calling the appropriate set or retrieve routine with a `NULL` item pointer. If an application changes a current default item attribute value, subsequent items created using `new_item()` will have the new default attribute value. The attributes of previously created items are not changed if a current default attribute value is changed.

Routine Name Index The following table lists each menus routine and the name of the manual page on which it is described.

| Menus Routine Name | Manual Page Name |
|-------------------------------|------------------------------------|
| <code>current_item</code> | <code>menu_item_current(3X)</code> |
| <code>free_item</code> | <code>menu_item_new(3X)</code> |
| <code>free_menu</code> | <code>menu_new(3X)</code> |
| <code>item_count</code> | <code>menu_items(3X)</code> |
| <code>item_description</code> | <code>menu_item_name(3X)</code> |
| <code>item_index</code> | <code>menu_item_current(3X)</code> |
| <code>item_init</code> | <code>menu_hook(3X)</code> |
| <code>item_name</code> | <code>menu_item_name(3X)</code> |
| <code>item_opts</code> | <code>menu_item_opts(3X)</code> |
| <code>item_opts_off</code> | <code>menu_item_opts(3X)</code> |
| <code>item_opts_on</code> | <code>menu_item_opts(3X)</code> |
| <code>item_term</code> | <code>menu_hook(3X)</code> |

| | |
|------------------|-----------------------|
| item_userptr | menu_item_userptr(3X) |
| item_value | menu_item_value(3X) |
| item_visible | menu_item_visible(3X) |
| menu_back | menu_attributes(3X) |
| menu_driver | menu_driver(3X) |
| menu_fore | menu_attributes(3X) |
| menu_format | menu_format(3X) |
| menu_grey | menu_attributes(3X) |
| menu_init | menu_hook(3X) |
| menu_items | menu_items(3X) |
| menu_mark | menu_mark(3X) |
| menu_opts | menu_opts(3X) |
| menu_opts_off | menu_opts(3X) |
| menu_opts_on | menu_opts(3X) |
| menu_pad | menu_attributes(3X) |
| menu_pattern | menu_pattern(3X) |
| menu_sub | menu_win(3X) |
| menu_term | menu_hook(3X) |
| menu_userptr | menu_userptr(3X) |
| menu_win | menu_win(3X) |
| new_item | menu_item_new(3X) |
| new_menu | menu_new(3X) |
| pos_menu_cursor | menu_cursor(3X) |
| post_menu | menu_post(3X) |
| scale_menu | menu_win(3X) |
| set_current_item | menu_item_current(3X) |
| set_item_init | menu_hook(3X) |
| set_item_opts | menu_item_opts(3X) |
| set_item_term | menu_hook(3X) |
| set_item_userptr | menu_item_userptr(3X) |
| set_item_value | menu_item_value(3X) |
| set_menu_back | menu_attributes(3X) |

| | |
|------------------|-----------------------|
| set_menu_fore | menu_attributes(3X) |
| set_menu_format | menu_format(3X) |
| set_menu_grey | menu_attributes(3X) |
| set_menu_init | menu_hook(3X) |
| set_menu_items | menu_items(3X) |
| set_menu_mark | menu_mark(3X) |
| set_menu_opts | menu_opts(3X) |
| set_menu_pad | menu_attributes(3X) |
| set_menu_pattern | menu_pattern(3X) |
| set_menu_sub | menu_win(3X) |
| set_menu_term | menu_hook(3X) |
| set_menu_userptr | menu_userptr(3X) |
| set_menu_win | menu_win(3X) |
| set_top_row | menu_item_current(3X) |
| top_row | menu_item_current(3X) |
| unpost_menu | menu_post(3X) |

RETURN VALUES

Routines that return pointers always return NULL on error. Routines that return an integer return one of the following:

| | |
|----------------|--|
| E_OK | The routine returned successfully. |
| E_SYSTEM_ERROR | System error. |
| E_BAD_ARGUMENT | An incorrect argument was passed to the routine. |
| E_POSTED | The menu is already posted. |
| E_CONNECTED | One or more items are already connected to another menu. |
| E_BAD_STATE | The routine was called from an initialization or termination function. |
| E_NO_ROOM | The menu does not fit within its subwindow. |
| E_NOT_POSTED | The menu has not been posted. |

| | |
|-------------------|---|
| E_UNKNOWN_COMMAND | An unknown request was passed to the menu driver. |
| E_NO_MATCH | The character failed to match. |
| E_NOT_SELECTABLE | The item cannot be selected. |
| E_NOT_CONNECTED | No items are connected to the menu. |
| E_REQUEST_DENIED | The menu driver could not process the request. |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

curses(3X), **attributes(5)**

NOTES

The header `<menu.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

| NAME | menu_userptr, set_menu_userptr – associate application data with menus | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lmenu -lcurses [<i>library</i> ..] #include <menu.h> char * menu_userptr(MENU * menu); int set_menu_userptr(MENU * menu, char * userptr);</pre> | | | | |
| DESCRIPTION | <p>Every menu has an associated user pointer that can be used to store relevant information. set_menu_userptr() sets the user pointer of <i>menu</i> .</p> <p>menu_userptr() returns the user pointer of <i>menu</i> .</p> | | | | |
| RETURN VALUES | <p>menu_userptr() returns NULL on error.</p> <p>set_menu_userptr() returns one of the following:</p> <pre>E_OK\011\011The routine returned successfully. E_SYSTEM_ERROR\011\011System error.</pre> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | curses(3X) , menus(3X) , attributes(5) | | | | |
| NOTES | The header <menu.h> automatically includes the headers <eti.h> and <curses.h> . | | | | |

| | |
|----------------------|--|
| NAME | menu_win, set_menu_win, set_menu_sub, menu_sub, scale_menu – menus window and subwindow association routines |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lmenu -lcurses [<i>library</i> ..] #include <menu.h> int set_menu_win(MENU * menu, WINDOW * win); WINDOW * menu_win(MENU * menu); int set_menu_sub(MENU * menu, WINDOW * sub); WINDOW * menu_sub(MENU * menu); int scale_window(MENU * menu, int * rows, int * cols); </pre> |
| DESCRIPTION | <p>set_menu_win() sets the window of <i>menu</i> to <i>win</i> . menu_win() returns a pointer to the window of <i>menu</i> .</p> <p>set_menu_sub() sets the subwindow of <i>menu</i> to <i>sub</i> . menu_sub() returns a pointer to the subwindow of <i>menu</i> .</p> <p>scale_window() returns the minimum window size necessary for the subwindow of <i>menu</i> . <i>rows</i> and <i>cols</i> are pointers to the locations used to return the values.</p> |
| RETURN VALUES | <p>Routines that return pointers always return NULL on error. Routines that return an integer return one of the following:</p> <pre> E_OK\011\011The routine returned successfully. E_SYSTEM_ERROR\011\011System error. E_BAD_ARGUMENT\011\011An incorrect argument was passed to \011\011the routine. E_POSTED\011\011The menu is already posted. E_NOT_CONNECTED\011\011No items are connected to the menu. </pre> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `curses(3X)`, `menus(3X)`, `attributes(5)`

NOTES The header `<menu.h>` automatically includes the headers `<eti.h>` and `<curses.h>`.

| | |
|----------------------|---|
| NAME | meta – enable/disable meta keys |
| SYNOPSIS | <pre>#include <curses.h> int meta(WINDOW *win, bool bf);</pre> |
| PARAMETERS | <p>win Is an ignored parameter.</p> <p>bf Is a Boolean expression.</p> |
| DESCRIPTION | <p>Whether a terminal returns 7 or 8 significant bits initially depends on the control mode of the terminal driver. The meta() function forces the number of bits to be returned by getch(3XC) to be 7 (if bf is FALSE) or 8 (if bf is TRUE).</p> <p>If the program handling the data can only pass 7-bit characters or strips the 8th bit, 8 bits cannot be handled.</p> <p>If the terminfo capabilities smm (meta_on) and rmm (meta_off) are defined for the terminal, smm is sent to the terminal when meta(win, TRUE) is called, and rmm is sent when meta(win, FALSE) is called.</p> <p>This function is useful when extending the non-text command set in applications where the META key is used.</p> |
| RETURN VALUES | On success, the meta() function returns OK . Otherwise, it returns ERR . |
| ERRORS | None. |
| SEE ALSO | getch(3XC) |

NAME mkdirp, rmdirp – create, remove directories in a path

SYNOPSIS

```
cc [
  flag
  ... ]
file
...
-lgen
[
  library
  ... ]
#include <libgen.h>

int mkdirp(const char * path, mode_t mode);

int rmdirp(char * dir, char * dir1);
```

DESCRIPTION

mkdirp() creates all the missing directories in the given *path* with the given *mode*. See **chmod(2)** for the values of *mode*.

rmdirp() removes directories in path *dir*. This removal starts at the end of the path and moves back toward the root as far as possible. If an error occurs, the remaining path is stored in *dir1*. **rmdirp()** returns a 0 only if it is able to remove every directory in the path.

RETURN VALUES

If a needed directory cannot be created, **mkdirp()** returns -1 and sets *errno* to one of the **mkdir()** error numbers. If all the directories are created, or existed to begin with, it returns zero.

EXAMPLES

EXAMPLE 1 Example of creating scratch directories.

```
/* create scratch directories */
if(mkdirp("/tmp/sub1/sub2/sub3", 0755) == -1) {
  \011fprintf(stderr, "cannot create directory");
  \011exit(1);
}
chdir("/tmp/sub1/sub2/sub3");
.
.
.
/* cleanup */
chdir("/tmp");
rmdirp("sub1/sub2/sub3");
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO `chmod(2)`, `mkdir(2)`, `rmdir(2)`, `malloc(3C)`, `attributes(5)`

NOTES `mkdirp()` uses `malloc(3C)` to allocate temporary space for the string.

`rmdirp()` returns `-2` if a “`.`” or “`..`” is in the path and `-3` if an attempt is made to remove the current directory. If an error occurs other than one of the above, `-1` is returned.

When compiling multithreaded applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multithreaded applications.

| NAME | mkfifo – create a new FIFO | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <sys/types.h> #include <sys/stat.h> int mkfifo(const char *<i>path</i>, mode_t <i>mode</i>);</pre> | | | | |
| DESCRIPTION | <p>The mkfifo() function creates a new FIFO special file named by the pathname pointed to by <i>path</i>. The mode of the new FIFO is initialized from <i>mode</i>. The file permission bits of the <i>mode</i> argument are modified by the process's file creation mask (see umask(2)). Bits other than the file permission bits in <i>mode</i> are ignored.</p> <p>The FIFO's owner id is set to the process's effective user ID. The FIFO's group ID is set to the process's effective group ID, or if the <code>S_ISGID</code> bit is set in the parent directory then the group ID of the FIFO is inherited from the parent directory.</p> <p>The mkfifo() function calls mknod(2) to make the file.</p> | | | | |
| RETURN VALUES | Upon successful completion, 0 is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | mkdir(1) , chmod(2) , exec(2) , mknod(2) , umask(2) , fs_ufs(4) , attributes(5) , stat(5) | | | | |

| | |
|----------------------|---|
| NAME | mkstemp – make a unique file name |
| SYNOPSIS | <pre>#include <stdlib.h> int mkstemp(char *template);</pre> |
| DESCRIPTION | <p>The mkstemp() function replaces the contents of the string pointed to by <i>template</i> by a unique file name, and returns a file descriptor for the file open for reading and writing. The function thus prevents any possible race condition between testing whether the file exists and opening it for use. The string in <i>template</i> should look like a file name with six trailing 'X's; mkstemp() replaces each 'X' with a character from the portable file name character set. The characters are chosen such that the resulting name does not duplicate the name of an existing file.</p> |
| RETURN VALUES | <p>Upon successful completion, mkstemp() returns an open file descriptor. Otherwise <code>-1</code> is returned if no suitable file could be created.</p> |
| ERRORS | <p>No errors are defined.</p> |
| USAGE | <p>It is possible to run out of letters.</p> <p>The mkstemp() function does not check to determine whether the file name part of <i>template</i> exceeds the maximum allowable file name length.</p> <p>The tmpfile(3S) function is preferred over this function.</p> <p>The mkstemp() function has a transitional interface for 64-bit file offsets. See 1f64(5).</p> |
| SEE ALSO | <p>getpid(2), open(2), tmpfile(3S), tmpnam(3S), 1f64(5), standards(5)</p> |

| NAME | mktemp – make a unique file name | | | | |
|----------------------|---|----------------|-----------------|----------|------|
| SYNOPSIS | #include <stdlib.h> char *mktemp(char *template); | | | | |
| DESCRIPTION | The mktemp() function replaces the contents of the string pointed to by <i>template</i> with a unique file name, and returns <i>template</i> . The string in <i>template</i> should look like a file name with six trailing 'X's; mktemp() will replace the 'X's with a character string that can be used to create a unique file name. Only 26 unique file names per thread can be created for each unique <i>template</i> . | | | | |
| RETURN VALUES | The mktemp() function will assign to <i>template</i> the empty string if it cannot create a unique name. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe | | | | |
| SEE ALSO | mkstemp(3C) , tmpfile(3S) , tmpnam(3S) , attributes(5) | | | | |

| | |
|--------------------|--|
| NAME | mktime – converts a tm structure to a calendar time |
| SYNOPSIS | <pre>#include <time.h> time_t mktime(struct tm *timeptr);</pre> |
| DESCRIPTION | <p>The mktime() function converts the time represented by the tm structure pointed to by <i>timeptr</i> into a calendar time (the number of seconds since 00:00:00 UTC, January 1, 1970).</p> <p>The tm structure contains the following members:</p> <pre>int tm_sec; /* seconds after the minute [0, 61] */ int tm_min; /* minutes after the hour [0, 59] */ int tm_hour; /* hour since midnight [0, 23] */ int tm_mday; /* day of the month [1, 31] */ int tm_mon; /* months since January [0, 11] */ int tm_year; /* years since 1900 */ int tm_wday; /* days since Sunday [0, 6] */ int tm_yday; /* days since January 1 [0, 365] */ int tm_isdst; /* flag for daylight savings time */</pre> <p>In addition to computing the calendar time, mktime() normalizes the supplied tm structure. The original values of the tm_wday and tm_yday components of the structure are ignored, and the original values of the other components are not restricted to the ranges indicated in the definition of the structure. On successful completion, the values of the tm_wday and tm_yday components are set appropriately, and the other components are set to represent the specified calendar time, but with their values forced to be within the appropriate ranges. The final value of tm_mday is not set until tm_mon and tm_year are determined.</p> <p>The tm_year member must be for year 1901 or later. Calendar times before 20:45:52 UTC, December 31, 1901 or after 03:14:07 UTC, January 19, 2038 cannot be represented. Portable applications should not try to create dates before 00:00:00 UTC, January 1, 1970 or after 00:00:00 UTC, January 1, 2038.</p> <p>The original values of the components may be either greater than or less than the specified range. For example, a tm_hour of -1 means 1 hour before midnight, tm_mday of 0 means the day preceding the current month, and tm_mon of -2 means 2 months before January of tm_year.</p> <p>If tm_isdst is positive, the original values are assumed to be in the alternate timezone. If it turns out that the alternate timezone is not valid for the computed calendar time, then the components are adjusted to the main timezone. Likewise, if tm_isdst is zero, the original values are assumed to be in the main timezone and are converted to the alternate timezone if the main</p> |

timezone is not valid. If `tm_isdst` is negative, **mktime()** attempts to determine whether the alternate timezone is in effect for the specified time.

Local timezone information is used as if **mktime()** had called **tzset()**. See **ctime(3C)**.

RETURN VALUES

The **mktime()** function returns the specified calendar time. If the calendar time cannot be represented, the function returns the value `(time_t)-1`.

USAGE

The **mktime()** function is MT-Safe in multithreaded applications, as long as no user-defined function directly modifies one of the following variables: `timezone`, `altzone`, `daylight`, and `tzname`. See **ctime(3C)**.

EXAMPLES

EXAMPLE 1 Sample code using **mktime()**.

What day of the week is July 4, 2001?

```
#include <stdio.h>
#include <time.h>
static char *const wday[] = {
    "Sunday", "Monday", "Tuesday", "Wednesday",
    "Thursday", "Friday", "Saturday", "-unknown-"
};
struct tm time_str;
/*...*/
time_str.tm_year = 2001 - 1900;
time_str.tm_mon = 7 - 1;
time_str.tm_mday = 4;
time_str.tm_hour = 0;
time_str.tm_min = 0;
time_str.tm_sec = 1;
time_str.tm_isdst = -1;
if (mktime(&time_str) == -1)
    time_str.tm_wday = 7;
printf("%s\n", wday[time_str.tm_wday]);
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |

SEE ALSO

ctime(3C), **getenv(3C)**, **TIMEZONE(4)**, **attributes(5)**

| | |
|----------------------------|--|
| NAME | mlock, munlock – lock or unlock pages in memory |
| SYNOPSIS | |
| Default | <pre>#include <sys/mman.h> int mlock(caddr_t addr, size_t len); int munlock(caddr_t addr, size_t len);</pre> |
| Standard-conforming | <pre>#include <sys/mman.h> int mlock(const void * addr, size_t len); int munlock(const void * addr, size_t len);</pre> |
| DESCRIPTION | <p>The mlock() function uses the mappings established for the address range [<i>addr</i>, <i>addr + len</i>) to identify pages to be locked in memory. If the page identified by a mapping changes, such as occurs when a copy of a writable <code>MAP_PRIVATE</code> page is made upon the first store, the lock will be transferred to the newly copied private page.</p> <p>The munlock() function removes locks established with mlock() .</p> <p>A given page may be locked multiple times by executing an mlock() through different mappings. That is, if two different processes lock the same page, then the page will remain locked until both processes remove their locks. However, within a given mapping, page locks do not nest – multiple mlock() operations on the same address in the same process will all be removed with a single munlock() . Of course, a page locked in one process and mapped in another (or visible through a different mapping in the locking process) is still locked in memory. This fact can be used to create applications that do nothing other than lock important data in memory, thereby avoiding page I/O faults on references from other processes in the system.</p> <p>If the mapping through which an mlock() has been performed is removed, an munlock() is implicitly performed. An munlock() is also performed implicitly when a page is deleted through file removal or truncation.</p> <p>Locks established with mlock() are not inherited by a child process after a fork() and are not nested.</p> <p>Attempts to mlock() more memory than a system-specific limit will fail.</p> |
| RETURN VALUES | <p>Upon successful completion, the mlock() and munlock() functions return 0 . Otherwise, no changes are made to any locks in the address space of the process, the functions return -1 and set <code>errno</code> to indicate the error.</p> |
| ERRORS | <p>The mlock() and munlock() functions will fail if:</p> |

- EINVAL** The *addr* argument is not a multiple of the page size as returned by `sysconf(3C)` .
- ENOMEM** Addresses in the range [*addr*, *addr + len*) are invalid for the address space of a process, or specify one or more pages which are not mapped.
- ENOSYS** The system does not support this memory locking interface.
- EPERM** The process's effective user ID is not super-user.
The `mlock()` function will fail if:
- EAGAIN** Some or all of the memory identified by the range [*addr*, *addr + len*) could not be locked because of insufficient system resources.

USAGE

Because of the impact on system resources, the use of `mlock()` and `munlock()` is restricted to the super-user.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`fork(2)` , `memcntl(2)` , `mmap(2)` , `plock(3C)` , `mlockall(3C)` , `sysconf(3C)` , `attributes(5)` , `standards(5)`

| | |
|----------------------|--|
| NAME | mlockall, munlockall – lock or unlock address space |
| SYNOPSIS | <pre>#include <sys/mman.h> int mlockall(int flags); int munlockall(void);</pre> |
| DESCRIPTION | <p>The mlockall() function locks in memory all pages mapped by an address space.</p> <p>The value of <i>flags</i> determines whether the pages to be locked are those currently mapped by the address space, those that will be mapped in the future, or both:</p> <pre>MCL_CURRENT Lock current mappings MCL_FUTURE Lock future mappings</pre> <p>If MCL_FUTURE is specified for mlockall(), mappings are locked as they are added to the address space (or replace existing mappings), provided sufficient memory is available. Locking in this manner is not persistent across the exec family of functions (see exec(2)).</p> <p>Mappings locked using mlockall() with any option may be explicitly unlocked with a munlock() call (see mlock(3C)).</p> <p>The munlockall() function removes address space locks and locks on mappings in the address space.</p> <p>All conditions and constraints on the use of locked memory that apply to mlock(3C) also apply to mlockall().</p> <p>Locks established with mlockall() are not inherited by a child process after a fork(2) call, and are not nested.</p> |
| RETURN VALUES | Upon successful completion, the mlockall() and munlockall() functions return 0. Otherwise, they return -1 and set errno to indicate the error. |
| ERRORS | <p>The mlockall() and munlockall() functions will fail if:</p> <p>EAGAIN Some or all of the memory in the address space could not be locked due to sufficient resources. This error condition applies to mlockall() only.</p> |

EINVAL The *flags* argument contains values other than `MCL_CURRENT` and `MCL_FUTURE`.

EPERM The process's effective user ID is not super-user.

USAGE

The `mlockall()` and `munlockall()` functions require super-user privileges.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`exec(2)`, `fork(2)`, `memcntl(2)`, `mmap(2)`, `plock(3C)`, `mlock(3C)`, `sysconf(3C)`, `attributes(5)`

| NAME | modf, modff – decompose floating-point number | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <math.h> double modf(double x, double * iptr); float modff(float x, float * iptr);</pre> | | | | |
| DESCRIPTION | The modf() and modff() functions break the argument <i>x</i> into integral and fractional parts, each of which has the same sign as the argument. The modf() function stores the integral part as a <code>double</code> in the object pointed to by <i>iptr</i> . The modff() function stores the integral part as a <code>float</code> in the object pointed to by <i>iptr</i> . | | | | |
| RETURN VALUES | <p>Upon successful completion, modf() and modff() return the signed fractional part of <i>x</i> .</p> <p>If <i>x</i> is NaN, NaN is returned and * <i>iptr</i> is set to NaN.</p> <p>If the correct value would cause underflow to 0.0, modf() returns 0 and <code>errno</code> may be set to <code>ERANGE</code> .</p> | | | | |
| ERRORS | <p>The modf() function may fail if:</p> <p>ERANGE The result underflows.</p> | | | | |
| USAGE | An application wishing to check for error situations should set <code>errno</code> to 0 before calling modf() . If <code>errno</code> is non-zero on return, or the return value is NaN, an error has occurred. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | frexp(3C) , isnan(3M) , ldexp(3C) , attributes(5) | | | | |

| | |
|--------------------|--|
| NAME | monitor – prepare process execution profile |
| SYNOPSIS | <pre>#include <mon.h> void monitor(int (*lowpc), int (*highpc)(), WORD *buffer, size_t bufsize, size_t nfunc);</pre> |
| DESCRIPTION | <p>The monitor() function is an interface to the profil(2) function and is called automatically with default parameters by any program created by the cc(1B) utility with the -p option specified. Except to establish further control over profiling activity, it is not necessary to explicitly call monitor().</p> <p>When used, monitor() is called at least at the beginning and the end of a program. The first call to monitor() initiates the recording of two different kinds of execution-profile information: execution-time distribution and function call count. Execution-time distribution data is generated by profil() and the function call counts are generated by code supplied to the object file (or files) by cc(1B) -p. Both types of information are collected as a program executes. The last call to monitor() writes this collected data to the output file <code>mon.out</code>.</p> <p>The name of the file written by monitor() is controlled by the environment variable <code>PROFDIR</code>. If <code>PROFDIR</code> does not exist, the file <code>mon.out</code> is created in the current directory. If <code>PROFDIR</code> exists but has no value, monitor() does no profiling and creates no output file. If <code>PROFDIR</code> is <code>dirname</code>, and monitor() is called automatically by compilation with <code>cc -p</code>, the file created is <code>dirname/pid.progname</code> where <code>progname</code> is the name of the program.</p> <p>The <code>lowpc</code> and <code>highpc</code> arguments are the beginning and ending addresses of the region to be profiled.</p> <p>The <code>buffer</code> argument is the address of a user-supplied array of <code>WORD</code> (defined in the header <code><mon.h></code>). The <code>buffer</code> argument is used by monitor() to store the histogram generated by profil() and the call counts.</p> <p>The <code>bufsize</code> argument identifies the number of array elements in <code>buffer</code>.</p> <p>The <code>nfunc</code> argument is the number of call count cells that have been reserved in <code>buffer</code>. Additional call count cells will be allocated automatically as they are needed.</p> <p>The <code>bufsize</code> argument should be computed using the following formula:</p> <pre>size_of_buffer = sizeof(struct hdr) + nfunc * sizeof(struct cnt) + ((highpc-lowpc)/BARSIZE) * sizeof(WORD) + sizeof(WORD) - 1 ; bufsize = (size_of_buffer / sizeof(WORD));</pre> <p>where:</p> <ul style="list-style-type: none"> ■ <code>lowpc</code>, <code>highpc</code>, <code>nfunc</code> are the same as the arguments to monitor(); |

- *BARSIZE* is the number of program bytes that correspond to each histogram bar, or cell, of the **profil()** buffer;
- the *hdr* and *cnt* structures and the type `WORD` are defined in the header `<mon.h>`.

The default call to **monitor()** is as follows:

```
monitor (&eprol, &etext, wbuf, wbufsz, 600);
```

where:

- *eprol* is the beginning of the user's program when linked with `cc -p` (see **end(3C)**);
- *etext* is the end of the user's program (see **end(3C)**);
- *wbuf* is an array of `WORD` with *wbufsz* elements;
- *wbufsz* is computed using the *bufsize* formula shown above with *BARSIZE* of 8;
- 600 is the number of call count cells that have been reserved in *buffer*.

These parameter settings establish the computation of an execution-time distribution histogram that uses **profil()** for the entire program, initially reserves room for 600 call count cells in *buffer*, and provides for enough histogram cells to generate significant distribution-measurement results. For more information on the effects of *bufsize* on execution-distribution measurements, see **profil(2)**.

EXAMPLES

EXAMPLE 1 Example to stop execution monitoring and write the results to a file.

To stop execution monitoring and write the results to a file, use the following:

```
monitor((int (*)())0, (int (*)())0, (WORD *)0, 0, 0);
```

Use `prof` to examine the results.

USAGE

Additional calls to **monitor()** after **main()** has been called and before **exit()** has been called will add to the function-call count capacity, but such calls will also replace and restart the **profil()** histogram computation.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

cc(1B), **profil(2)**, **end(3C)**, **attributes(5)**, **prof(5)**

| | | | | | | | |
|----------------------|--|----------|--|----------|---|------------|--|
| NAME | move, wmove – move cursor in window | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int move(int y, int x); int wmove(WINDOW * win, int y, int x);</pre> | | | | | | |
| PARAMETERS | <table><tr><td>y</td><td>Is the y (row) coordinate of the position of the cursor in the window.</td></tr><tr><td>x</td><td>Is the x (column) coordinate of the position of the cursor in the window.</td></tr><tr><td>win</td><td>Is a pointer to the window in which the cursor is to be written.</td></tr></table> | y | Is the y (row) coordinate of the position of the cursor in the window. | x | Is the x (column) coordinate of the position of the cursor in the window. | win | Is a pointer to the window in which the cursor is to be written. |
| y | Is the y (row) coordinate of the position of the cursor in the window. | | | | | | |
| x | Is the x (column) coordinate of the position of the cursor in the window. | | | | | | |
| win | Is a pointer to the window in which the cursor is to be written. | | | | | | |
| DESCRIPTION | The move() function moves the logical cursor (for <code>stdscr</code>) to the position specified by <i>y</i> (row) and <i>x</i> (column), where the upper left corner of the window is row 0, column 0. The wmove() function performs the same action, but moves the cursor in the window specified by <i>win</i> . The physical cursor will not move until after a call to refresh(3XC) or doupdate(3XC) . | | | | | | |
| RETURN VALUES | On success, these functions return <code>OK</code> . Otherwise, they return <code>ERR</code> . | | | | | | |
| ERRORS | None. | | | | | | |
| SEE ALSO | doupdate(3XC) | | | | | | |

| | |
|--------------------|--|
| NAME | mp, mp_madd, mp_msub, mp_mult, mp_mdiv, mp_mcmp, mp_min, mp_mout, mp_pow, mp_gcd, mp_rpow, mp_itom, mp_xtom, mp_mtox, mp_mfree – multiple precision integer arithmetic |
| SYNOPSIS | <pre>cc [flag ...] file ... -lmp [library ...] #include <mp.h> void mp_madd(MINT * a, MINT * b, MINT * c); void mp_msub(MINT * a, MINT * b, MINT * c); void mp_mult(MINT * a, MINT * b, MINT * c); void mp_mdiv(MINT * a, MINT * b, MINT * q, MINT * r); int mp_mcmp(MINT * a, MINT * b); int mp_min(MINT * a); void mp_mout(MINT * a); void mp_pow(MINT * a, MINT * b, MINT * c, MINT * d); void mp_gcd(MINT * a, MINT * b, MINT * c); void mp_rpow(MINT * a, short n, MINT * b); int mp_msqrt(MINT * a, MINT * b, MINT * r); void mp_sdiv(MINT * a, short n, MINT * q, short * r); MINT * mp_itom(short n); MINT * mp_xtom(char * a); char * mp_mtox(MINT * a); void mp_mfree(MINT * a);</pre> |
| DESCRIPTION | <p>These routines perform arithmetic on integers of arbitrary length. The integers are stored using the defined type <code>MINT</code>. Pointers to a <code>MINT</code> should be initialized using the function <code>mp_itom(n)</code>, which sets the initial value to <code>n</code>. Alternatively, <code>mp_xtom(a)</code> may be used to initialize a <code>MINT</code> from a string.</p> |

of hexadecimal digits. `mp_mfree(a)` may be used to release the storage allocated by the `mp_itom(a)` and `mp_xtom(a)` routines.

The `mp_madd(a , b , c)`, `mp_msub(a , b , c)` and `mp_mult(a , b , c)` functions assign to their third arguments the sum, difference, and product, respectively, of their first two arguments. The `mp_mdiv(a , b , q , r)` function assigns the quotient and remainder, respectively, to its third and fourth arguments. The `mp_sdiv(a , n , q , r)` function is similar to `mp_mdiv(a , b , q , r)` except that the divisor is an ordinary integer. The `mp_msqrt(a , b , r)` function produces the square root and remainder of its first argument. The `mp_mcmp(a , b)` function compares the values of its arguments and returns 0 if the two values are equal, a value greater than 0 if the first argument is greater than the second, and a value less than 0 if the second argument is greater than the first. The `mp_rpow(a , n , b)` function raises `a` to the `n`th power and assigns this value to `b`. The `mp_pow(a , b , c , d)` function raises `a` to the `b`th power, reduces the result modulo `c` and assigns this value to `d`. The `mp_min(a)` and `mp_mout(a)` functions perform decimal input and output. The `mp_gcd(a , b , c)` function finds the greatest common divisor of the first two arguments, returning it in the third argument. The `mp_mtox(a)` function provides the inverse of `mp_xtom(a)`. To release the storage allocated by `mp_mtox(a)`, use `free()` (see `malloc(3C)`).

Use the `-lmp` loader option to obtain access to these functions.

FILES

`/usr/lib/libmp.a`
`/usr/lib/libmp.so`

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`exp(3M)`, `malloc(3C)`, `libmp(4)`, `attributes(5)`

DIAGNOSTICS

Illegal operations and running out of memory produce messages and core images.

WARNINGS

The function `pow()` exists in both `libmp` and `libm` with widely differing semantics. This is why `libmp.so.2` exists. `libmp.so.1` exists solely for reasons of backward compatibility, and should not be used otherwise. Use the `mp_*()` functions instead. See `libmp(4)`.

NAME mq_close – close a message queue

SYNOPSIS

```
cc [ flag... ] file... -lrt [ library... ]
#include <mqqueue.h>

int mq_close(mqd_t mqdes);
```

DESCRIPTION

The **mq_close()** function removes the association between the message queue descriptor, *mqdes*, and its message queue. The results of using this message queue descriptor after successful return from this **mq_close()**, and until the return of this message queue descriptor from a subsequent **mq_open(3R)**, are undefined.

If the process (or thread) has successfully attached a notification request to the message queue via this *mqdes*, this attachment is removed and the message queue is available for another process to attach for notification.

RETURN VALUES

Upon successful completion, **mq_close()** returns 0; otherwise, the function returns -1 and sets *errno* to indicate the error condition.

ERRORS

The **mq_close()** function will fail if:

EBADF The *mqdes* argument is an invalid message queue descriptor.

ENOSYS The **mq_open()** function is not supported by the system.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

mq_notify(3R), **mq_open(3R)**, **mq_unlink(3R)**, **attributes(5)**, **mqqueue(5)**

NOTES

Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned -1 and set *errno* to ENOSYS.

| NAME | mq_getattr – get message queue attributes | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <mqueue.h> int mq_getattr(mqd_t mqdes, struct mq_attr *mqstat);</pre> | | | | |
| DESCRIPTION | <p>The <i>mqdes</i> argument specifies a message queue descriptor. The mq_getattr() function is used to get status information and attributes of the message queue and the open message queue description associated with the message queue descriptor. The results are returned in the <i>mq_attr</i> structure referenced by the <i>mqstat</i> argument.</p> <p>Upon return, the following members will have the values associated with the open message queue description as set when the message queue was opened and as modified by subsequent mq_setattr(3R) calls:</p> <p><i>mq_flags</i> message queue flags</p> <p>The following attributes of the message queue are returned as set at message queue creation:</p> <p><i>mq_maxmsg</i> maximum number of messages</p> <p><i>mq_msgsize</i> maximum message size</p> <p><i>mq_curmsgs</i> number of messages currently on the queue.</p> | | | | |
| RETURN VALUES | Upon successful completion, the mq_getattr() function returns 0. Otherwise, the function returns -1 and sets <i>errno</i> to indicate the error. | | | | |
| ERRORS | <p>The mq_getattr() function will fail if:</p> <p>EBADF The <i>mqdes</i> argument is not a valid message queue descriptor.</p> <p>ENOSYS The mq_getattr() function is not supported by the system.</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | msgctl(2) , msgget(2) , msgrcv(2) , msgsnd(2) , mq_open(3R) , mq_send(3R) , mq_setattr(3R) , attributes(5) , mqueue(5) | | | | |

NOTES

Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned `-1` and set `errno` to `ENOSYS`.

| | |
|--------------------|--|
| NAME | mq_notify - notify process (or thread) that a message is available on a queue |
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <mqueue.h> int mq_notify(mqd_t mqdes, const struct sigevent *notification);</pre> |
| DESCRIPTION | <p>The mq_notify() function provides an asynchronous mechanism for processes to receive notice that messages are available in a message queue, rather than synchronously blocking (waiting) in mq_receive(3R).</p> <p>If <i>notification</i> is not <code>NULL</code>, this function registers the calling process to be notified of message arrival at an empty message queue associated with the message queue descriptor, <i>mqdes</i>. The notification specified by <i>notification</i> will be sent to the process when the message queue transitions from empty to non-empty. At any time, only one process may be registered for notification by a specific message queue. If the calling process or any other process has already registered for notification of message arrival at the specified message queue, subsequent attempts to register for that message queue will fail.</p> <p>The <i>notification</i> argument points to a structure that defines both the signal to be generated and how the calling process will be notified upon I/O completion. If <i>notification->sigev_notify</i> is <code>SIGEV_NONE</code>, then no signal will be posted upon I/O completion, but the error status and the return status for the operation will be set appropriately. If <i>notification->sigev_notify</i> is <code>SIGEV_SIGNAL</code>, then the signal specified in <i>notification->sigev_signo</i> will be sent to the process. If the <code>SA_SIGINFO</code> flag is set for that signal number, then the signal will be queued to the process and the value specified in <i>notification->sigev_value</i> will be the <code>si_value</code> component of the generated signal (see siginfo(5)).</p> <p>If <i>notification</i> is <code>NULL</code> and the process is currently registered for notification by the specified message queue, the existing registration is removed. The message queue is then available for future registration.</p> <p>When the notification is sent to the registered process, its registration is removed. The message queue is then be available for registration.</p> <p>If a process has registered for notification of message arrival at a message queue and some processes is blocked in mq_receive(3R) waiting to receive a message when a message arrives at the queue, the arriving message will be received by the appropriate mq_receive(3R), and no notification will be sent to the registered process. The resulting behavior is as if the message queue remains empty, and this notification will not be sent until the next arrival of a message at this queue.</p> <p>Any notification registration is removed if the calling process either closes the message queue or exits.</p> |

RETURN VALUES

Upon successful completion, **mq_notify()** returns 0; otherwise, it returns -1 and sets `errno` to indicate the error.

ERRORS

The **mq_notify()** function will fail if:

- EBADF** The *mqdes* argument is not a valid message queue descriptor.
- EBUSY** A process is already registered for notification by the message queue.
- ENOSYS** The **mq_notify()** function is not supported by the system.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

mq_close(3R), **mq_open(3R)**, **mq_receive(3R)**, **mq_send(3R)**, **attributes(5)**, **mqueue(5)**, **siginfo(5)**, **signal(5)**

NOTES

Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned -1 and set `errno` to `ENOSYS`.

| | | | | | | | |
|--------------------|--|----------|--|----------|--|--------|---|
| NAME | mq_open – open a message queue | | | | | | |
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <mqueue.h> mqd_t mq_open(const char *name, int oflag, /* unsigned long mode, mq_attr attr */ ...);</pre> | | | | | | |
| DESCRIPTION | <p>The mq_open() function establishes the connection between a process and a message queue with a message queue descriptor. It creates a open message queue description that refers to the message queue, and a message queue descriptor that refers to that open message queue description. The message queue descriptor is used by other functions to refer to that message queue.</p> <p>The <i>name</i> argument points to a string naming a message queue. The <i>name</i> argument must conform to the construction rules for a path-name. If <i>name</i> is not the name of an existing message queue and its creation is not requested, mq_open() fails and returns an error. The first character of <i>name</i> must be a slash (/) character and the remaining characters of <i>name</i> cannot include any slash characters. For maximum portability, <i>name</i> should include no more than 14 characters, but this limit is not enforced.</p> <p>The <i>oflag</i> argument requests the desired receive and/or send access to the message queue. The requested access permission to receive messages or send messages is granted if the calling process would be granted read or write access, respectively, to a file with the equivalent permissions.</p> <p>The value of <i>oflag</i> is the bitwise inclusive OR of values from the following list. Applications must specify exactly one of the first three values (access modes) below in the value of <i>oflag</i>:</p> <table border="0"> <tr> <td style="padding-right: 20px;">O_RDONLY</td> <td>Open the message queue for receiving messages. The process can use the returned message queue descriptor with mq_receive(3R), but not mq_send(3R). A message queue may be open multiple times in the same or different processes for receiving messages.</td> </tr> <tr> <td style="padding-right: 20px;">O_WRONLY</td> <td>Open the queue for sending messages. The process can use the returned message queue descriptor with mq_send(3R) but not mq_receive(3R). A message queue may be open multiple times in the same or different processes for sending messages.</td> </tr> <tr> <td style="padding-right: 20px;">O_RDWR</td> <td>Open the queue for both receiving and sending messages. The process can use any of the functions allowed for O_RDONLY and O_WRONLY. A message queue may be open</td> </tr> </table> | O_RDONLY | Open the message queue for receiving messages. The process can use the returned message queue descriptor with mq_receive(3R) , but not mq_send(3R) . A message queue may be open multiple times in the same or different processes for receiving messages. | O_WRONLY | Open the queue for sending messages. The process can use the returned message queue descriptor with mq_send(3R) but not mq_receive(3R) . A message queue may be open multiple times in the same or different processes for sending messages. | O_RDWR | Open the queue for both receiving and sending messages. The process can use any of the functions allowed for O_RDONLY and O_WRONLY. A message queue may be open |
| O_RDONLY | Open the message queue for receiving messages. The process can use the returned message queue descriptor with mq_receive(3R) , but not mq_send(3R) . A message queue may be open multiple times in the same or different processes for receiving messages. | | | | | | |
| O_WRONLY | Open the queue for sending messages. The process can use the returned message queue descriptor with mq_send(3R) but not mq_receive(3R) . A message queue may be open multiple times in the same or different processes for sending messages. | | | | | | |
| O_RDWR | Open the queue for both receiving and sending messages. The process can use any of the functions allowed for O_RDONLY and O_WRONLY. A message queue may be open | | | | | | |

multiple times in the same or different processes for sending messages.

Any combination of the remaining flags may additionally be specified in the value of *oflag*:

O_CREAT This option is used to create a message queue, and it requires two additional arguments: *mode*, which is of type *mode_t*, and *attr*, which is pointer to a *mq_attr* structure. If the pathname, *name*, has already been used to create a message queue that still exists, then this flag has no effect, except as noted under **O_EXCL** (see below). Otherwise, a message queue is created without any messages in it.

The user ID of the message queue is set to the effective user ID of process, and the group ID of the message queue is set to the effective group ID of the process. The file permission bits are set to the value of *mode*, and modified by clearing all bits set in the file mode creation mask of the process (see **umask(2)**).

If *attr* is non-NULL and the calling process has the appropriate privilege on *name*, the message queue *mq_maxmsg* and *mq_msgsiz*e attributes are set to the values of the corresponding members in the *mq_attr* structure referred to by *attr*. If *attr* is non-NULL, but the calling process does not have the appropriate privilege on *name*, the **mq_open()** function fails and returns an error without creating the message queue.

O_EXCL If both **O_EXCL** and **O_CREAT** are set, **mq_open()** will fail if the message queue *name* exists. The check for the existence of the message queue and the creation of the message queue if it does not exist are atomic with respect to other processes executing **mq_open()** naming the same *name* with both **O_EXCL** and **O_CREAT** set. If **O_EXCL** and **O_CREAT** are not set, the result is undefined.

O_NONBLOCK The setting of this flag is associated with the open message queue description and determines whether a **mq_send(3R)** or **mq_receive(3R)** waits for resources or messages that are not currently available, or fails with *errno* set to **EAGAIN**. See **mq_send(3R)** and **mq_receive(3R)** for details.

RETURN VALUES

Upon successful completion, **mq_open()** returns a message queue descriptor; otherwise the function returns $(mqd_t)-1$ and sets *errno* to indicate the error condition.

ERRORS

The **mq_open()** function will fail if:

| | |
|---------------------|--|
| EACCESS | The message queue exists and the permissions specified by <i>oflag</i> are denied, or the message queue does not exist and permission to create the message queue is denied. |
| EEXIST | <code>O_CREAT</code> and <code>O_EXCL</code> are set and the named message queue already exists. |
| EINTR | The mq_open() operation was interrupted by a signal. |
| EINVAL | The mq_open() operation is not supported for the given name, or <code>O_CREAT</code> was specified in <i>oflag</i> , the value of <i>attr</i> is not <code>NULL</code> , and either <code>mq_maxmsg</code> or <code>mq_msgsize</code> was less than or equal to zero. |
| EMFILE | The number of open message queue descriptors in this process exceeds <code>MQ_OPEN_MAX</code> , or the number of open file descriptors in this process exceeds <code>OPEN_MAX</code> . |
| ENAMETOOLONG | The length of the <i>name</i> string exceeds <code>PATH_MAX</code> , or a pathname component is longer than <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect. |
| ENFILE | Too many message queues are currently open in the system. |
| ENOENT | <code>O_CREAT</code> is not set and the named message queue does not exist. |
| ENOSPC | There is insufficient space for the creation of the new message queue. |
| ENOSYS | The mq_open() function is not supported by the system. |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO `exec(2)`, `exit(2)`, `umask(2)`, `mq_close(3R)`, `mq_receive(3R)`, `mq_send(3R)`, `mq_setattr(3R)`, `mq_unlink(3R)`, `sysconf(3C)`, `attributes(5)`, `mqueue(5)`

NOTES Due to the manner in which message queues are implemented, they should not be considered secure and should not be used in security-sensitive applications.

Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned `-1` and set `errno` to `ENOSYS`.

| | |
|----------------------|--|
| NAME | mq_receive – receive a message from a message queue |
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <mqueue.h> ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len, unsigned int *msg_prio);</pre> |
| DESCRIPTION | <p>The mq_receive() function is used to receive the oldest of the highest priority message(s) from the message queue specified by <i>mqdes</i>. If the size of the buffer in bytes, specified by <i>msg_len</i>, is less than the <code>mq_msgsize</code> member of the message queue, the function fails and returns an error. Otherwise, the selected message is removed from the queue and copied to the buffer pointed to by <i>msg_ptr</i>.</p> <p>If <i>msg_prio</i> is not NULL, the priority of the selected message is stored in the location referenced by <i>msg_prio</i>.</p> <p>If the specified message queue is empty and <code>O_NONBLOCK</code> is not set in the message queue description associated with <i>mqdes</i>, (see mq_open(3R) and mq_setattr(3R)), mq_receive() blocks, waiting until a message is enqueued on the message queue, or until mq_receive() is interrupted by a signal. If more than one process (or thread) is waiting to receive a message when a message arrives at an empty queue, then the process of highest priority that has been waiting the longest is selected to receive the message. If the specified message queue is empty and <code>O_NONBLOCK</code> is set in the message queue description associated with <i>mqdes</i>, no message is removed from the queue, and mq_receive() returns an error.</p> |
| RETURN VALUES | Upon successful completion, mq_receive() returns the length of the selected message in bytes and the message is removed from the queue. Otherwise, no message is removed from the queue, the function returns a value of <code>-1</code> , and sets <code>errno</code> to indicate the error condition. |
| ERRORS | <p>The mq_receive() function will fail if:</p> <p>EAGAIN <code>O_NONBLOCK</code> was set in the message description associated with <i>mqdes</i>, and the specified message queue is empty.</p> <p>EBADF The <i>mqdes</i> argument is not a valid message queue descriptor open for reading.</p> <p>EMSGSIZE The specified message buffer size, <i>msg_len</i>, is less than the message size member of the message queue.</p> <p>EINTR The mq_receive() function operation was interrupted by a signal.</p> |

ENOSYS The **mq_receive()** function is not supported by the system.
The **mq_receive()** function may fail if:

EBADMSG A data corruption problem with the message has been detected.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

mq_open(3R), **mq_send(3R)**, **mq_setattr(3R)**, **attributes(5)**, **mqueue(5)**

NOTES

Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned `-1` and set `errno` to `ENOSYS`.

| | |
|----------------------|--|
| NAME | mq_send – send a message to a message queue |
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <mqueue.h> int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned int msg_prio);</pre> |
| DESCRIPTION | <p>The mq_send() function adds the message pointed to by the argument <i>msg_ptr</i> to the message queue specified by <i>mqdes</i>. The <i>msg_len</i> argument specifies the length of the message in bytes pointed to by <i>msg_ptr</i>. The value of <i>msg_len</i> is less than or equal to the <i>mq_msgsize</i> attribute of the message queue, or mq_send() fails.</p> <p>If the specified message queue is not full, mq_send() behaves as if the message is inserted into the message queue at the position indicated by the <i>msg_prio</i> argument. A message with a larger numeric value of <i>msg_prio</i> is inserted before messages with lower values of <i>msg_prio</i>. A message will be inserted after other messages in the queue, if any, with equal <i>msg_prio</i>. The value of <i>msg_prio</i> must be greater than zero and less than or equal to <code>MQ_PRIO_MAX</code>.</p> <p>If the specified message queue is full and <code>O_NONBLOCK</code> is not set in the message queue description associated with <i>mqdes</i> (see mq_open(3R) and mq_setattr(3R)), mq_send() blocks until space becomes available to enqueue the message, or until mq_send() is interrupted by a signal. If more than one thread is waiting to send when space becomes available in the message queue, then the thread of the highest priority which has been waiting the longest is unblocked to send its message. Otherwise, it is unspecified which waiting thread is unblocked. If the specified message queue is full and <code>O_NONBLOCK</code> is set in the message queue description associated with <i>mqdes</i>, the message is not queued and mq_send() returns an error.</p> |
| RETURN VALUES | Upon successful completion, mq_send() returns 0; otherwise, no message is enqueued, the function returns -1, and <code>errno</code> is set to indicate the error. |
| ERRORS | <p>The mq_send() function will fail if:</p> <p>EAGAIN The <code>O_NONBLOCK</code> flag is set in the message queue description associated with <i>mqdes</i>, and the specified message queue is full.</p> <p>EBADF The <i>mqdes</i> argument is not a valid message queue descriptor open for writing.</p> <p>EINTR A signal interrupted the call to mq_send()</p> <p>EINVAL The value of <i>msg_prio</i> was outside the valid range.</p> |

EMSGSIZE The specified message length, *msg_len*, exceeds the message size attribute of the message queue.

ENOSYS The **mq_send()** function is not supported by the system.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

mq_open(3R), **mq_receive(3R)**, **mq_setattr(3R)**, **sysconf(3C)**, **attributes(5)**, **mqueue(5)**

NOTES

Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned `-1` and set `errno` to `ENOSYS`.

| NAME | mq_setattr – set/get message queue attributes | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <mqueue.h> int mq_setattr(mqd_t mqdes, const struct mq_attr *mqstat, struct mq_attr *omqstat);</pre> | | | | |
| DESCRIPTION | <p>The mq_setattr() function is used to set attributes associated with the open message queue description referenced by the message queue descriptor specified by <i>mqdes</i>.</p> <p>The message queue attributes corresponding to the following members defined in the <i>mq_attr</i> structure are set to the specified values upon successful completion of mq_setattr():</p> <p><i>mq_flags</i> The value of this member is either 0 or O_NONBLOCK. The values of <i>mq_maxmsg</i>, <i>mq_msgsize</i>, and <i>mq_curmsgs</i> are ignored by mq_setattr().</p> <p>If <i>omqstat</i> is non-NULL, mq_setattr() stores, in the location referenced by <i>omqstat</i>, the previous message queue attributes and the current queue status. These values are the same as would be returned by a call to mq_getattr() at that point.</p> | | | | |
| RETURN VALUES | Upon successful completion, mq_setattr() returns 0 and the attributes of the message queue will have been changed as specified. Otherwise, the message queue attributes are unchanged, and the function returns -1 and sets <i>errno</i> to indicate the error. | | | | |
| ERRORS | <p>The mq_setattr() function will fail if:</p> <p>EBADF The <i>mqdes</i> argument is not a valid message queue descriptor.</p> <p>ENOSYS The mq_setattr() function is not supported by the system.</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | mq_getattr(3R) , mq_open(3R) , mq_receive(3R) , mq_send(3R) , attributes(5) , mqueue(5) | | | | |
| NOTES | Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned -1 and set <i>errno</i> to ENOSYS. | | | | |

NAME mq_unlink – remove a message queue

SYNOPSIS

```
cc [ flag... ] file... -lrt [ library... ]
#include <mqueue.h>

int mq_unlink(const char *name);
```

DESCRIPTION

The **mq_unlink()** function removes the message queue named by the pathname *name*. After a successful call to **mq_unlink()** with *name*, a call to **mq_open(3R)** with *name* fails if the flag `O_CREAT` is not set in *flags*. If one or more processes have the message queue open when **mq_unlink()** is called, destruction of the message queue is postponed until all references to the message queue have been closed. Calls to **mq_open(3R)** to re-create the message queue may fail until the message queue is actually removed. However, the **mq_unlink()** call need not block until all references have been closed; it may return immediately.

RETURN VALUES

Upon successful completion, **mq_unlink()** returns 0; otherwise, the named message queue is not changed by this function call, the function returns -1 and sets `errno` to indicate the error.

ERRORS

The **mq_unlink()** function will fail if:

- EACCESS** Permission is denied to unlink the named message queue.
- ENAMETOOLONG** The length of the *name* string exceeds `PATH_MAX`, or a pathname component is longer than `NAME_MAX` while `_POSIX_NO_TRUNC` is in effect.
- ENOENT** The named message queue, *name*, does not exist.
- ENOSYS** **mq_unlink()** is not supported by the system.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **mq_close(3R)**, **mq_open(3R)**, **attributes(5)**, **mqueue(5)**

NOTES

Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned -1 and set `errno` to `ENOSYS`.

| | |
|----------------------|--|
| NAME | msync – synchronize memory with physical storage |
| SYNOPSIS | <pre>#include <sys/mman.h> int msync(void *addr, size_t len, int flags);</pre> |
| DESCRIPTION | <p>The msync() function writes all modified copies of pages over the range [<i>addr</i>; <i>addr + len</i>) to the underlying hardware, or invalidates any copies so that further references to the pages will be obtained by the system from their permanent storage locations. The permanent storage for a modified <code>MAP_SHARED</code> mapping is the file the page is mapped to; the permanent storage for a modified <code>MAP_PRIVATE</code> mapping is its swap area.</p> <p>The <i>flags</i> argument is a bit pattern built from the following values:</p> <p>MS_ASYNC perform asynchronous writes</p> <p>MS_SYNC perform synchronous writes</p> <p>MS_INVALIDATE invalidate mappings</p> <p>If <i>flags</i> is <code>MS_ASYNC</code> or <code>MS_SYNC</code>, the function synchronizes the file contents to match the current contents of the memory region.</p> <ul style="list-style-type: none"> ■ All write references to the memory region made prior to the call are visible by subsequent read operations on the file. ■ All writes to the same portion of the file prior to the call may or may not be visible by read references to the memory region. ■ Unmodified pages in the specified range are not written to the underlying hardware. <p>If <i>flags</i> is <code>MS_ASYNC</code>, the function may return immediately once all write operations are scheduled; if <i>flags</i> is <code>MS_SYNC</code>, the function does not return until all write operations are completed.</p> <p>If <i>flags</i> is <code>MS_INVALIDATE</code>, the function synchronizes the contents of the memory region to match the current file contents.</p> <ul style="list-style-type: none"> ■ All writes to the mapped portion of the file made prior to the call are visible by subsequent read references to the mapped memory region. ■ All write references prior to the call, by any process, to memory regions mapped to the same portion of the file using <code>MAP_SHARED</code>, are visible by read references to the region. <p>If msync() causes any write to the file, then the file's <code>st_ctime</code> and <code>st_mtime</code> fields are marked for update.</p> |
| RETURN VALUES | Upon successful completion, msync() returns 0; otherwise, it returns -1 and sets <code>errno</code> to indicate the error. |

ERRORS

The **msync()** function will fail if:

- EBUSY** Some or all of the addresses in the range [*addr*, *addr + len*) are locked and **MC_SYNC** with the **MS_INVALIDATE** option is specified.
- EINVAL** The *addr* argument is not a multiple of the page size as returned by **sysconf(3C)**.
The *flags* argument is not some combination of **MS_ASYNC** and **MS_INVALIDATE**.
- EIO** An I/O error occurred while reading from or writing to the file system.
- ENOMEM** Addresses in the range [*addr*, *addr + len*) are outside the valid range for the address space of a process, or specify one or more pages that are not mapped.
- EPERM** **MS_INVALIDATE** was specified and one or more of the pages is locked in memory.

USAGE

The **msync()** function should be used by programs that require a memory object to be in a known state, for example in building transaction facilities.

Normal system activity can cause pages to be written to disk. Therefore, there are no guarantees that **msync()** is the only control over when pages are or are not written to disk.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

memcntl(2), **mmap(2)**, **sysconf(3C)**, **attributes(5)**

| | |
|--------------------|--|
| NAME | mtmalloc, malloc, calloc, realloc, free, mallocctl – MT hot memory allocator |
| SYNOPSIS | <pre>#include <mtmalloc.h> cc -o a.out -pthread -lmtmalloc void * malloc(size_t size); void * calloc(size_t nelem, size_t elsize); void * realloc(void * ptr, size_t size); void free(void * ptr); void mallocctl(int cmd, long value);</pre> |
| DESCRIPTION | <p>malloc() and free() provide a simple general-purpose memory allocation package that is suitable for use in high performance multithreaded applications. The suggested use of this library is in multithreaded applications; it can be used for single threaded applications, but there is no advantage in doing so. This library cannot be dynamically loaded via dlopen() during runtime because there must be only one manager of the process heap.</p> <p>malloc() returns a pointer to a block of at least <code>size</code> bytes suitably aligned for any use.</p> <p>The argument to free() is a pointer to a block previously allocated by malloc(), calloc() or realloc(). After free() is performed this space is available for further allocation. If <code>ptr</code> is a null pointer, no action occurs.</p> <p>Undefined results will occur if the space assigned by malloc() is overrun or if a random number is handed to free(). A freed pointer that is passed to free() will send a SIGABRT signal to the calling process. This behavior is controlled by mallocctl().</p> <p>calloc() allocates a zero-initialized space for an array of <code>nelem</code> elements of size <code>elsize</code>.</p> <p>realloc() changes the size of the block pointed to by <code>ptr</code> to <code>size</code> bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes. If <code>ptr</code> is NULL, realloc() behaves like malloc() for the specified size. If <code>size</code> is 0 and <code>ptr</code> is not a null pointer, the object pointed to is freed.</p> <p>After possible pointer coercion, each allocation routine returns a pointer to a space that is suitably aligned for storage of any type of object.</p> <p>malloc(), realloc() and calloc() will fail if there is not enough available memory.</p> <p>mallocctl() controls the behavior of the <code>malloc</code> library. The options fall into two general classes, debugging options and performance options.</p> |

| | |
|----------------|--|
| MTDOUBLEFREE | Allows double <code>free</code> of a pointer. Setting <i>value</i> to 1 means yes and 0 means no. The default behavior of double <code>free</code> results in a core dump. |
| MTDEBUGPATTERN | Writes misaligned data into the buffer after <code>free()</code> . When the buffer is reallocated, the contents are verified to ensure that there was no access to the buffer after the <code>free</code> . If the buffer has been dirtied, a <code>SIGABRT</code> signal is delivered to the process. Setting <i>value</i> to 1 means yes and 0 means no. The default behavior is to <i>not</i> write misaligned data. The pattern used is <code>0xdeadbeef</code> . Use of this option results in a performance penalty. |
| MTINITBUFFER | Writes misaligned data into the newly allocated buffer. This option is useful for detecting some accesses before initialization. Setting <i>value</i> to 1 means yes and 0 means no. The default behavior is to <i>not</i> write misaligned data to the newly allocated buffer. The pattern used is <code>0xbaddcafe</code> . Use of this option results in a performance penalty. |
| MTCHUNKSIZE | This option changes the size of allocated memory when a pool has exhausted all available memory in the buffer. Increasing this value allocates more memory for the application. A substantial performance gain can occur because the library makes fewer calls to the OS for more memory. Acceptable number <i>value</i> s are between 9 and 256; the default value is 9. This value is multiplied by 8192. |

RETURN VALUES

If there is no available memory, `malloc()`, `realloc()`, and `calloc()` return a null pointer. When `realloc()` returns `NULL`, the block pointed to by *ptr* is left intact. If *size*, *nelem*, or *elsize* is 0, a unique pointer to the arena is returned.

ERRORS

If `malloc()`, `calloc()` or `realloc()` return unsuccessfully, `errno` will be set to indicate the following:

ENOMEM *size* bytes of memory exceeds the physical limits of your system, and cannot be allocated.

EAGAIN There is not enough memory available *at this time* to allocate `size` bytes of memory; but the application could try again later.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

`brk(2)`, `getrlimit(2)`, `bsdmalloc(3X)`, `dlopen(3X)`, `malloc(3C)`, `malloc(3X)`, `mapmalloc(3X)`, `watchmalloc(3X)`, `attributes(5)`, `signal(5)`

WARNINGS

Undefined results will occur if the size requested for a block of memory exceeds the maximum size of a process's heap. This information may be obtained using `getrlimit()`.

NOTES

Comparative Features of `malloc(3C)`, `bsdmalloc(3X)`, `malloc(3X)`, and `mtmalloc(3T)`.

- The `bsdmalloc(3X)` routines afford better performance, but are space-inefficient.
- The `malloc(3X)` routines are space-efficient, but have slower performance.
- The standard, fully SCD-compliant `malloc` routines are a trade-off between performance and space-efficiency.
- The `mtmalloc` routines provide fast, concurrent `malloc` implementation that is space-inefficient.

`free()` does not set `errno`.

NAME mutex – concepts relating to mutual exclusion locks

DESCRIPTION

| FUNCTION | ACTION |
|-----------------------|--------------------------|
| mutex_init | Initialize a mutex. |
| mutex_destroy | Destroy a mutex. |
| mutex_lock | Lock a mutex. |
| mutex_trylock | Attempt to lock a mutex. |
| mutex_unlock | Unlock a mutex. |
| pthread_mutex_init | Initialize a mutex. |
| pthread_mutex_destroy | Destroy a mutex. |
| pthread_mutex_lock | Lock a mutex. |
| pthread_mutex_trylock | Attempt to lock a mutex. |
| pthread_mutex_unlock | Unlock a mutex. |

Mutual exclusion locks (mutexes) prevent multiple threads from simultaneously executing critical sections of code which access shared data (that is, mutexes are used to serialize the execution of threads). All mutexes must be global. A successful call to acquire a mutex will cause another thread that is also trying to lock the same mutex to block until the owner thread unlocks the mutex.

Mutexes can synchronize threads within the same process or in other processes. Mutexes can be used to synchronize threads between processes if the mutexes are allocated in writable memory and shared among the cooperating processes (see `mmap(2)`), and have been initialized for this task.

Initialization

Mutexes are either intra-process or inter-process, depending upon the argument passed implicitly or explicitly to the initialization of that mutex. A statically allocated mutex does not need to be explicitly initialized; by default, a statically allocated mutex is initialized with all zeros and its scope is set to be within the calling process.

For inter-process synchronization, a mutex needs to be allocated in memory shared between these processes. Since the memory for such a mutex must be allocated dynamically, the mutex needs to be explicitly initialized with the appropriate attribute that indicates inter-process use.

**Locking and
Unlocking**

A critical section of code is enclosed by a call to lock the mutex and the call to unlock the mutex to protect it from simultaneous access by multiple threads. Only one thread at a time may possess mutually exclusive access to the critical section of code that is enclosed by the mutex-locking call and the

mutex-unlocking call, whether the mutex's scope is intra-process or inter-process. A thread calling to lock the mutex either gets exclusive access to the code starting from the successful locking until its call to unlock the mutex, or it waits until the mutex is unlocked by the thread that locked it.

Mutexes have ownership, unlike semaphores. Only the thread that locked a mutex, (that is, the owner of the mutex), should unlock it.

If a thread waiting for a mutex receives a signal, upon return from the signal handler, the thread resumes waiting for the mutex as if there was no interrupt.

Caveats

Mutexes are almost like data – they can be embedded in data structures, files, dynamic or static memory, and so forth. Hence, they are easy to introduce into a program. However, too many mutexes can degrade performance and scalability of the application. Because too few mutexes can hinder the concurrency of the application, they should be introduced with care. Also, incorrect usage (such as recursive calls, or violation of locking order, and so forth) can lead to deadlocks, or worse, data inconsistencies.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`mmap(2)`, `shmop(2)`, `mutex_destroy(3T)`, `mutex_init(3T)`, `mutex_lock(3T)`, `mutex_trylock(3T)`, `mutex_unlock(3T)`, `pthread_mutex_destroy(3T)`, `pthread_mutex_init(3T)`, `pthread_mutex_lock(3T)`, `pthread_mutex_trylock(3T)`, `pthread_mutex_unlock(3T)`, `pthread_create(3T)`, `pthread_mutexattr_init(3T)`, `attributes(5)`, `standards(5)`

NOTES

In the current implementation of threads, `pthread_mutex_lock()`, `pthread_mutex_unlock()`, `mutex_lock()`, `mutex_unlock()`, `pthread_mutex_trylock()`, and `mutex_trylock()` do not validate the mutex type. Therefore, an uninitialized mutex or a mutex with an invalid type does not return `EINVAL`. Interfaces for mutexes with an invalid type have unspecified behavior.

By default, if multiple threads are waiting for a mutex, the order of acquisition is undefined.

`USYNC_THREAD` does not support multiple mappings to the same logical synch object. If you need to `mmap()` a synch object to different locations within the same address space, then the synch object should be initialized as a

shared object USYNC_PROCESS for Solaris, and PTHREAD_PROCESS_PRIVATE for POSIX.

| | |
|--------------------|--|
| NAME | mutex_init, mutex_destroy, mutex_lock, mutex_trylock, mutex_unlock – mutual exclusion locks |
| SYNOPSIS | <pre>cc - mt [<i>flag</i> ...] <i>file</i> ...[<i>library</i> ...] #include <thread.h> #include <synch.h> int mutex_init(mutex_t * mp, int type, void * arg); int mutex_lock(mutex_t * mp); int mutex_trylock(mutex_t * mp); int mutex_unlock(mutex_t * mp); int mutex_destroy(mutex_t * mp);</pre> |
| DESCRIPTION | <p>Mutual exclusion locks (mutexes) prevent multiple threads from simultaneously executing critical sections of code which access shared data (that is, mutexes are used to serialize the execution of threads). All mutexes must be global. A successful call for a mutex lock by way of mutex_lock() will cause another thread that is also trying to lock the same mutex to block until the owner thread unlocks it by way of mutex_unlock() . Threads within the same process or within other processes can share mutexes.</p> <p>Mutexes can synchronize threads within the same process or in other processes. Mutexes can be used to synchronize threads between processes if the mutexes are allocated in writable memory and shared among the cooperating processes (see mmap(2)), and have been initialized for this task.</p> |
| Initialize | <p>Mutexes are either intra-process or inter-process, depending upon the argument passed implicitly or explicitly to the initialization of that mutex. A statically allocated mutex does not need to be explicitly initialized; by default, a statically allocated mutex is initialized with all zeros and its scope is set to be within the calling process.</p> <p>For inter-process synchronization, a mutex needs to be allocated in memory shared between these processes. Since the memory for such a mutex must be allocated dynamically, the mutex needs to be explicitly initialized using mutex_init() .</p> <p>The mutex_init() function initializes the mutex referenced by <i>mp</i> with the type specified by <i>type</i> . Upon successful initialization the state of the mutex</p> |

becomes initialized and unlocked. No current type uses *arg* although a future type may specify additional behavior parameters by way of *arg*. *type* may be one of the following:

| | |
|----------------------|--|
| USYNC_THREAD | The mutex can synchronize threads only in this process. <i>arg</i> is ignored. |
| USYNC_PROCESS | The mutex can synchronize threads in this process and other processes. <i>arg</i> is ignored. The object initialized with this attribute must be allocated in memory shared between processes, either in System V shared memory (see <code>shmop(2)</code>) or in memory mapped to a file (see <code>mmap(2)</code>). If the object is not allocated in such shared memory, it will not be shared between processes. |
| USYNC_PROCESS_ROBUST | The mutex can synchronize threads in this process and other processes robustly. At the time of process death, if the lock is held by the process, it is unlocked. The next owner of this mutex will acquire it with an error return of <code>EOWNERDEAD</code> . Note that the application must always check the return code from <code>mutex_lock()</code> for a mutex of this type. The new owner of this mutex should then attempt to make the state protected by the mutex consistent, since this state could have been left inconsistent when the last owner died. If the new owner is able to make the state consistent, it should re-initialize the mutex and then unlock the mutex. If the new owner is not able to make the state consistent, for whatever reason, it should not re-initialize the mutex, but should just unlock the mutex. If the latter event occurs, all waiters will be woken up and all subsequent calls to <code>mutex_lock()</code> will fail in acquiring the mutex with an error code of <code>ENOTRECOVERABLE</code> . mutex can be made consistent by un-initializing the mutex (<code>mutex_destroy()</code>) and re-initializing it (<code>mutex_init()</code>). If the process which got the lock with <code>EOWNERDEAD</code> died, the next owner will get the lock with an error return of <code>EOWNERDEAD</code> . <i>arg</i> is ignored. The object initialized with this attribute must be allocated in memory shared between processes, either in System V shared memory (see <code>shmop(2)</code>) or in memory mapped to a file (see <code>mmap(2)</code>) and memory must be zeroed |

before initialization. All the processes interested in the robust lock must call **mutex_init()** at least once to register robust mutex with the system and potentially initialize it. If the object is not allocated in such shared memory, it will not be shared between processes. If **mutex_init()** is called on a previously initialized mutex **mutex_init()** will not re-initialize the mutex.

Initializing mutexes can also be accomplished by allocating in zeroed memory (default), in which case, a type of `USYNC_THREAD` is assumed. The same mutex must not be simultaneously initialized by multiple threads. A mutex lock must not be re-initialized while in use by other threads. If default mutex attributes are used, the macro `DEFAULTMUTEX` can be used to initialize mutexes that are statically allocated.

Default mutex initialization (intra-process):

```
mutex_t mp;  
mutex_init(&mp, NULL, NULL);
```

OR

```
mutex_init(&mp, USYNC_THREAD, NULL);
```

OR

```
mutex_t mp = DEFAULTMUTEX;
```

OR

```
mutex_t mp;  
mp = calloc(1, sizeof (mutex_t));
```

OR

```
mutex_t mp;  
mp = malloc(sizeof (mutex_t));  
memset(mp, 0, sizeof (mutex_t));
```

Customized mutex initialization (inter-process):

```
mutex_init(&mp, USYNC_PROCESS, NULL);
```

Customized mutex initialization (inter-process):

```
mutex_init(&mp, USYNC_PROCESS_ROBUST, NULL);
```

Lock and Unlock

A critical section of code is enclosed by a the call to lock the mutex and the call to unlock the mutex to protect it from simultaneous access by multiple threads. Only one thread at a time may possess mutually exclusive access to the critical section of code that is enclosed by the mutex-locking call and the mutex-unlocking call, whether the mutex's scope is intra-process or inter-process. A thread calling to lock the mutex either gets exclusive access to the code starting from the successful locking until its call to unlock the mutex, or it waits until the mutex is unlocked by the thread that locked it.

Mutexes have ownership, unlike semaphores. Although any thread, within the scope of a mutex, can get an unlocked mutex and lock access to the same critical section of code, only the thread that locked a mutex should unlock it.

If a thread waiting for a mutex receives a signal, upon return from the signal handler, the thread resumes waiting for the mutex as if there was no interrupt. A mutex protects code, not data; therefore, strongly bind a mutex with the data by putting both within the same structure, or at least within the same procedure.

A call to **mutex_lock()** locks the mutex object referenced by *mp* . If the mutex is already locked, the calling thread blocks until the mutex is freed; this will return with the mutex object referenced by *mp* in the locked state with the calling thread as its owner. If the current owner of a mutex tries to relock the mutex, it will result in deadlock.

mutex_trylock() is the same as **mutex_lock()** , respectively, except that if the mutex object referenced by *mp* is locked (by any thread, including the current thread), the call returns immediately with an error.

mutex_unlock() are called by the owner of the mutex object referenced by *mp* to release it. The mutex must be locked and the calling thread must be the one that last locked the mutex (the owner). If there are threads blocked on the mutex object referenced by *mp* when **mutex_unlock()** is called, the *mp* is freed, and the scheduling policy will determine which thread gets the mutex. If the calling thread is not the owner of the lock, no error status is returned, and the behavior of the program is undefined.

Destroy **mutex_destroy()** destroys the mutex object referenced by *mp* ; the mutex object becomes uninitialized. The space used by the destroyed mutex variable is not freed. It needs to be explicitly reclaimed.

RETURN VALUES If successful, these functions return 0 . Otherwise, an error number is returned.

ERRORS These functions may fail if:

EFAULT *mp* points to an illegal address.
The **mutex_init()** function will fail if:

EINVAL The value specified by `type` is invalid.
The **mutex_init()** function will fail for `USYNC_PROCESS_ROBUST` type mutex if:

EBUSY The mutex pointed to by *mp* was already initialized. An attempt to re-initialize a mutex previously initialized, but not yet destroyed.
The **mutex_trylock()** function will fail if:

EBUSY The mutex pointed to by *mp* was already locked.
The **mutex_lock()** or **mutex_trylock()** functions will fail for `USYNC_PROCESS_ROBUST` type mutex if:

EOWNERDEAD The last owner of this mutex died while holding the mutex. This mutex is now owned by the caller. The caller must now attempt to make the state protected by the mutex consistent. If it is able to cleanup the state, then it should re-initialize the mutex (see **mutex_init()**)and unlock the mutex. Subsequent calls to **mutex_lock()** will behave normally, as before. If the caller is not able to cleanup the state, the mutex should not be re-initialized, it should be unlocked. Subsequent calls to **mutex_lock()** will fail to acquire the mutex, with the error code, `ENOTRECOVERABLE` . If the owner who got the lock with `EOWNERDEAD` died, the next owner will get the lock with `EOWNERDEAD` .

ELOCKUNMAPPED The last owner of this mutex unmaped the mutex while holding the mutex. This mutex is now owned by the caller. The caller must now attempt to make the state protected by the mutex consistent. If it is able to cleanup the state, then it should re-initialize the mutex unlock the mutex. See **mutex_init(3T)** . Subsequent calls to **mutex_lock()** will behave normally, as before. If the caller is not able to cleanup the state, the

mutex should not be re-initialized. Subsequent calls to **mutex_lock()** will fail to acquire the mutex with the error code, **ENOTRECOVERABLE**.

ENOTRECOVERABLE

The mutex trying to be acquired is protecting state which has been left irrecoverable by the mutex's last owner, which died while holding the lock. The mutex has not been acquired. This condition can occur when the lock was previously acquired with **EOWNERDEAD** or **ELOCKUNMAPPED** and the owner was not able to cleanup the state and unlocked the mutex without making the mutex consistent.

EXAMPLES**Single Gate**

The following example uses one global mutex as a gate-keeper to permit each thread exclusive sequential access to the code within the user-defined function "change_global_data." This type of synchronization will protect the state of shared data, but it also prohibits parallelism.

```

/* cc thisfile.c -lthread */
#define _REENTRANT
#include <stdio.h>
#include <thread.h>
#define NUM_THREADS 12
void *change_global_data(void *); /* for thr_create() */
main(int argc, char * argv[]) \011{
    int i=0;
    for (i=0; i< NUM_THREADS; i++) \011{
        thr_create(NULL, 0, change_global_data, NULL, 0, NULL);
    }
    while ((thr_join(NULL, NULL, NULL) == 0));
}

void * change_global_data(void *null) \011{
    static mutex_t \011Global_mutex;
    static int \011Global_data = 0;
    mutex_lock(&Global_mutex);
    Global_data++;
    sleep(1);
    printf("%d is global data\
", Global_data); \011 \011
    mutex_unlock(&Global_mutex); \011
    return NULL;
}

```

**Multiple Instruction
Single Data**

The previous example, the mutex, the code it owns, and the data it protects was enclosed in one function. The next example uses C++ features to accommodate many functions that use just one mutex to protect one data:

```

/* CC thisfile.c -lthread use C++ to compile*/

#define _REENTRANT
#include <stdlib.h>
#include <stdio.h>
#include <thread.h>
#include <errno.h>
#include <iostream.h>
#define NUM_THREADS 16
void *change_global_data(void *); /* for thr_create() */

class Mutected {
private:
    static mutex_t\011 Global_mutex;
    static int Global_data;
public:
    static int add_to_global_data(void);
    static int subtract_from_global_data(void);
};

int Mutected::Global_data = 0;
mutex_t Mutected::Global_mutex;

int Mutected::add_to_global_data() {
    mutex_lock(&Global_mutex);\011
    Global_data++;
    mutex_unlock(&Global_mutex);\011
    return Global_data;
}

int Mutected::subtract_from_global_data() {
    mutex_lock(&Global_mutex);\011
    Global_data--;
    mutex_unlock(&Global_mutex);
    return Global_data;
}

void
main(int argc, char * argv[]) {
    int i=0;
    for (i=0; i< NUM_THREADS; i++) {
        thr_create(NULL, 0, change_global_data, NULL, 0, NULL);
    }
    while ((thr_join(NULL, NULL, NULL) == 0));
}

void * change_global_data(void *)\011{
    static int switcher = 0;
    if ((switcher++ % 3) == 0) /* one-in-three threads subtracts */
        cout << Mutected::subtract_from_global_data() << endl;\011
    else
        cout << Mutected::add_to_global_data() << endl;\011\011
}

```

Interprocess Locking

```

    return NULL;
}

```

A mutex can protect data that is shared among processes. The mutex would need to be initialized as `USYNC_PROCESS`. One process initializes the process-shared mutex and writes it to a file to be mapped into memory by all cooperating processes (see `mmap(2)`). Afterwards, other independent processes can run the same program (whether concurrently or not) and share mutex-protected data.

```

/* cc thisfile.c -lthread */
/* To execute, run the command line "a.out 0 & a.out 1" */

#define _REENTRANT
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <thread.h>
#define INTERPROCESS_FILE "ipc-sharedfile"
#define NUM_ADDTHREADS 12
#define NUM_SUBTRACTTHREADS 10
#define INCREMENT '0'
#define DECREMENT '1'
typedef struct {
    mutex_t      Interprocess_mutex;
    int          Interprocess_data;
} buffer_t;
buffer_t *buffer;

void *add_interprocess_data(), *subtract_interprocess_data();
void create_shared_memory(), test_argv();
int zeroed[sizeof(buffer_t)];
int ipc_fd, i=0;

void
main(int argc, char * argv[]){
    test_argv(argv[1]);

    switch (*argv[1]) {
    case INCREMENT:
        create_shared_memory();
        ipc_fd = open(INTERPROCESS_FILE, O_RDWR);
        buffer = (buffer_t *)mmap(NULL, sizeof(buffer_t),
            PROT_READ|PROT_WRITE, MAP_SHARED, ipc_fd, 0);
        buffer->Interprocess_data = 0;
        mutex_init(&buffer->Interprocess_mutex, USYNC_PROCESS, 0);
        for (i=0; i< NUM_ADDTHREADS; i++)
            thr_create(NULL, 0, add_interprocess_data, argv[1],
                0, NULL);
        break;

```

```

        case DECREMENT:
            while((ipc_fd = open(INTERPROCESS_FILE, O_RDWR)) == -1)
                sleep(1);
            buffer = (buffer_t *)mmap(NULL, sizeof(buffer_t),
                PROT_READ|PROT_WRITE, MAP_SHARED, ipc_fd, 0);
            for (i=0; i< NUM_SUBTRACTTHREADS; i++)
                thr_create(NULL, 0, subtract_interprocess_data, argv[1],
                    0, NULL);
            break;
        } /* end switch */

        while ((thr_join(NULL,NULL,NULL) == 0));
    } /* end main */

void *add_interprocess_data(char argv_1[]){
    mutex_lock(&buffer->Interprocess_mutex);
    buffer->Interprocess_data++;
    sleep(2);
    printf("%d is add-interprocess data, and %c is argv1\
",
        buffer->Interprocess_data, argv_1[0]);
    mutex_unlock(&buffer->Interprocess_mutex);
    return NULL;
}

void *subtract_interprocess_data(char argv_1[])\011{
    mutex_lock(&buffer->Interprocess_mutex);
    buffer->Interprocess_data--;
    sleep(2);
    printf("%d is subtract-interprocess data, and %c is argv1\
",
        buffer->Interprocess_data, argv_1[0]);
    mutex_unlock(&buffer->Interprocess_mutex);
    return NULL;
}

void create_shared_memory(){
    int i;
    ipc_fd = creat(INTERPROCESS_FILE, O_CREAT|O_RDWR );
    for (i=0; i<sizeof(buffer_t); i++){
        zeroed[i] = 0;
        write(ipc_fd, &zeroed[i],2);
    }
    close(ipc_fd);
    chmod(INTERPROCESS_FILE, S_IRWXU|S_IRWXG|S_IRWXO);
}

void test_argv(char argv1[]) {
    if (argv1 == NULL) {
        printf("use 0 as arg1 for initial process\
\\
or use 1 as arg1 for the second process\
");
        exit(NULL);
    }
}

```

**Solaris Interprocess
Robust Locking**

In this example, run the command line

```
a.out 0 & a.out 1
```

A mutex can protect data that is shared among processes robustly. The mutex would need to be initialized as `USYNC_PROCESS_ROBUST`. One process initializes the robust process-shared mutex and writes it to a file to be mapped into memory by all cooperating processes (see `mmap(2)`). Afterwards, other independent processes can run the same program (whether concurrently or not) and share mutex-protected data.

The following example shows how to use a `USYNC_PROCESS_ROBUST` type mutex.

```
/* cc thisfile.c -lthread */
/* To execute, run the command line "a.out & a.out 1" */
#include <sys/types.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <stdio.h>
#include <thread.h>
#define INTERPROCESS_FILE "ipc-sharedfile"
typedef struct {
    mutex_t    Interprocess_mutex;
    int        Interprocess_data;
} buffer_t;
buffer_t *buffer;
int make_date_consistent();
void create_shared_memory();
int zeroed[sizeof(buffer_t)];
int ipc_fd, i=0;
main(int argc, char * argv[]) {
    int rc;
    if (argc > 1) {
        while((ipc_fd = open(INTERPROCESS_FILE, O_RDWR)) == -1)
            sleep(1);
        buffer = (buffer_t *)mmap(NULL, sizeof(buffer_t),
            PROT_READ|PROT_WRITE, MAP_SHARED, ipc_fd, 0);
        mutex_init(&buffer->Interprocess_mutex,
            USYNC_PROCESS_ROBUST, 0);
    } else {
        create_shared_memory();
        ipc_fd = open(INTERPROCESS_FILE, O_RDWR);
        buffer = (buffer_t *)mmap(NULL, sizeof(buffer_t),
            PROT_READ|PROT_WRITE, MAP_SHARED, ipc_fd, 0);
        buffer->Interprocess_data = 0;
        mutex_init(&buffer->Interprocess_mutex,
            USYNC_PROCESS_ROBUST, 0);
    }
    for(;;) {
        rc = mutex_lock(&buffer->Interprocess_mutex);
```

```

        switch (rc) {
        case EOWNERDEAD:
            /* lock acquired.
             * last owner died holding the lock, try to make
             * the state associated with the mutex consistent.
             * If so, make the robust lock consistent by
             * re-initializing it.
             */
            if (make_data_consistent())
                mutex_init(&buffer->Interprocess_mutex,
                           USYNC_PROCESS_ROBUST, 0);
            mutex_unlock(&buffer->Interprocess_mutex);
        case ENOTRECOVERABLE:
            /* lock not acquired.
             * last owner got the mutex with EOWNERDEAD
             * mutex is not consistent (and data?),
             * so return from here
             */
            exit(1);
            break;
        case 0:
            /* no error - data is consistent */
            /* do something with data */
            mutex_unlock(&buffer->Interprocess_mutex);
            break;
        }
    }
} /* end main */
void create_shared_memory() {
    int i;
    ipc_fd = creat(INTERPROCESS_FILE, O_CREAT|O_RDWR );
    for (i=0; i<sizeof(buffer_t); i++) {
        zeroed[i] = 0;
        write(ipc_fd, &zeroed[i], 2);
    }
    close(ipc_fd);
    chmod(INTERPROCESS_FILE, S_IRWXU|S_IRWXG|S_IRWXO);
}

/* return 1 if able to make data consistent, otherwise 0. */
int make_data_consistent () {
    buffer->Interprocess_data = 0;
    return (1);
}

```

Dynamically Allocated Mutexes

The following example allocates and frees memory in which a mutex is embedded.

```

struct record {
    int field1;
    int field2;
    mutex_t m;
} *r;

```

```

r = malloc(sizeof(struct record));
mutex_init(&r->m, USYNC_THREAD, NULL);
/*
 * The fields in this record are accessed concurrently
 * by acquiring the embedded lock.
 */

```

The thread execution in this example is as follows:

Thread 1 executes:

Thread 2 executes:

```

...
mutex_lock(&r->m);
r->field1++;
mutex_unlock(&r->m);
...
...
mutex_lock(&r->m);
localvar = r->field1;
mutex_unlock(&r->m);
...

```

Later, when a thread decides to free the memory pointed to by *r*, the thread should call `mutex_destroy()` on the mutexes in this memory.

In the following example, the main thread can do a `thr_join()` on both of the above threads. If there are no other threads using the memory in *r*, the main thread can now safely free *r*:

```

for (i = 0; i < 2; i++)
    thr_join(0, 0, 0);
mutex_destroy(&r->m); /* first destroy mutex */
free(r);             /* Then free memory */

```

If the mutex is not destroyed, the program could have memory leaks.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`mmap(2)`, `shmop(2)`, `mutex(3T)`, `attributes(5)`, `standards(5)`

NOTES

Currently, the only supported policy is `SCHED_OTHER`. In Solaris, under the `SCHED_OTHER` policy, there is no established order in which threads are unblocked.

In the current implementation of threads, `mutex_lock()`, `mutex_unlock()`, and `mutex_trylock()` do not validate the mutex type. Therefore, an uninitialized mutex or a mutex with an invalid type does not return `EINVAL`. Interfaces for mutexes with an invalid type have unspecified behavior.

Since Uninitialized mutexes which are allocated locally may contain junk data. Such mutexes need to be initialized using `mutex_init()`.

By default, if multiple threads are waiting for a mutex, the order of acquisition is undefined.

| | | | | | | | | | |
|----------------------|--|---------------|--|---------------|---|---------------|--|---------------|---|
| NAME | mvcur – move the cursor | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int mvcur(int oldrow, int oldcol, int newrow, int newcol);</pre> | | | | | | | | |
| PARAMETERS | <table><tr><td><i>oldrow</i></td><td>Is the row from which cursor is to be moved.</td></tr><tr><td><i>oldcol</i></td><td>Is the column from which cursor is to be moved.</td></tr><tr><td><i>newrow</i></td><td>Is the row to which cursor is to be moved.</td></tr><tr><td><i>newcol</i></td><td>Is the column to which cursor is to be moved.</td></tr></table> | <i>oldrow</i> | Is the row from which cursor is to be moved. | <i>oldcol</i> | Is the column from which cursor is to be moved. | <i>newrow</i> | Is the row to which cursor is to be moved. | <i>newcol</i> | Is the column to which cursor is to be moved. |
| <i>oldrow</i> | Is the row from which cursor is to be moved. | | | | | | | | |
| <i>oldcol</i> | Is the column from which cursor is to be moved. | | | | | | | | |
| <i>newrow</i> | Is the row to which cursor is to be moved. | | | | | | | | |
| <i>newcol</i> | Is the column to which cursor is to be moved. | | | | | | | | |
| DESCRIPTION | <p>The mvcur() function is a low-level function used only outside of X/Open Curses when the program has to deal directly with the <code>terminfo</code> database to handle certain terminal capabilities. The use of appropriate X/Open Curses functions is recommended in all other situations, so that X/Open Curses can track the cursor.</p> <p>The mvcur() function moves the cursor from the location specified by <i>oldrow</i> and <i>oldcol</i> to the location specified by <i>newrow</i> and <i>newcol</i>. A program using this function must keep track of the current cursor position.</p> | | | | | | | | |
| RETURN VALUES | On success, the mvcur() function returns <code>OK</code> . Otherwise, it returns <code>ERR</code> . | | | | | | | | |
| ERRORS | None. | | | | | | | | |

| | | | | | | | |
|----------------------|--|-------------------|--|---------------------|--|---------------------|---|
| NAME | mvderwin – map area of parent window to subwindow | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int mvderwin(WINDOW *win, int par_y, int par_x);</pre> | | | | | | |
| PARAMETERS | <table><tr><td><i>win</i></td><td>Is a pointer to the window to be mapped.</td></tr><tr><td><i>par_y</i></td><td>Is the y (row) coordinate of the placement of the upper left corner of window relative to the parent window.</td></tr><tr><td><i>par_x</i></td><td>Is the x (column) coordinate of the placement of the upper left corner of the window relative to the parent window.</td></tr></table> | <i>win</i> | Is a pointer to the window to be mapped. | <i>par_y</i> | Is the y (row) coordinate of the placement of the upper left corner of window relative to the parent window. | <i>par_x</i> | Is the x (column) coordinate of the placement of the upper left corner of the window relative to the parent window. |
| <i>win</i> | Is a pointer to the window to be mapped. | | | | | | |
| <i>par_y</i> | Is the y (row) coordinate of the placement of the upper left corner of window relative to the parent window. | | | | | | |
| <i>par_x</i> | Is the x (column) coordinate of the placement of the upper left corner of the window relative to the parent window. | | | | | | |
| DESCRIPTION | <p>The mvderwin() function defines a mapped area of <i>win</i>'s parent window that is the same size as <i>win</i> and has its upper left corner at position <i>par_y</i>, <i>par_x</i> of the parent window.</p> <p>Whenever <i>win</i> is refreshed, its contents are updated to match those of the mapped area and any reference to characters in <i>win</i> is treated as a reference to corresponding characters in the mapped area.</p> | | | | | | |
| RETURN VALUES | On success, the mvderwin() function returns OK. Otherwise, it returns ERR. | | | | | | |
| ERRORS | None. | | | | | | |
| SEE ALSO | delwin(3XC) , derwin(3XC) | | | | | | |

| | |
|--------------------|---|
| NAME | mvprintw, mvwprintw, printw, vw_printw, vwprintw, wprintw – write formatted output to window |
| SYNOPSIS | <pre>#include <curses.h> int mvprintw(int y, int x, char * fmt [,arg...]); int mvwprintw(WINDOW * win, int y, int x, char * fmt [,arg...]); int printw(char * fmt [,arg...]); int vwprintw(WINDOW * win, char * fmt, void * arglist); int vw_printw(WINDOW * win, char * fmt, void * arglist); int wprintw(WINDOW * win, char * fmt[,arg...]);</pre> |
| PARAMETERS | <p>y Is the y (row) coordinate position of the string's placement in the window.</p> <p>x Is the x (column) coordinate position of the string's placement in the window.</p> <p>fmt [,arg...] Is a printf() format string where <i>arg</i> is zero or more parameters used to satisfy the printf() string.</p> <p>win Is a pointer to the window in which the string is to be written.</p> <p>fmt, arglist Is a vprintf() format string where <i>arglist</i> is a pointer to a list of parameters. The vwprintw() function requires a variable parameter list as defined in <code><varargs.h></code> . The vw_printw() function requires a variable parameter list as defined in <code><stdarg.h></code> .</p> |
| DESCRIPTION | <p>These functions are functionally equivalent to printf(3S) . Their effect is similar to using sprintf(3S) to format the string and then using waddstr(3XC) to add that string to a window.</p> <p>With printw() and wprintw() , the string is written to <code>stdscr</code> and <i>win</i> , respectively. The mvprintw() and mvwprintw() functions position the cursor as specified in <code>stdscr</code> or <i>win</i> , respectively, and then call printw() .</p> <p>The vwprintw() and vw_printw() functions are similar to wprintw() but use a pointer to a variable parameter list as defined by either <code><varargs.h></code> or <code><stdarg.h></code> . Each application must include the appropriate header.</p> |

RETURN VALUES

On success, these functions return `OK` . Otherwise, they return `ERR` .

ERRORS

None.

SEE ALSO

`addnstr(3XC)` , `mvscanw(3XC)` , `printf(3S)` , `sprintf(3S)`

| | |
|--------------------|---|
| NAME | mvscanw, mvwscanw, scanw, vw_scanw, vwscanw, wscanw – read formatted input from window |
| SYNOPSIS | <pre>#include <curses.h> int mvscanw(int y, int x, char * fmt[,arg...]); int mvwscanw(WINDOW * win, int y, int x, char * fmt[,arg...]); int scanw(char * fmt [,arg...]); int vwscanw(WINDOW * win, char * fmt, void * arglist); int vw_scanw(WINDOW * win, char * fmt, void * arglist); int wscanw(WINDOW * win, char * fmt [,arg...]);</pre> |
| PARAMETERS | <p>y Is the y (row) coordinate of the position of the character to be read.</p> <p>x Is the x (column) coordinate of the position of the character to be read.</p> <p>fmt [,arg...] <i>fmt</i> is a vwscanw() format string; <i>arg</i> is zero or more parameters used to satisfy the scanf() string.</p> <p>win Is a pointer to the window in which the character is to be read.</p> <p>fmt, arglist <i>fmt</i> is a scanf() format string; <i>arglist</i> is a pointer to zero or more parameters used to satisfy the scanf() string. The vwprintw() function requires a variable parameter list as defined in <code><varargs.h></code>. The vw_printw() function requires a variable parameter list as defined in <code><stdarg.h></code>.</p> |
| DESCRIPTION | <p>These functions are functionally equivalent to scanf(3S). Characters are read from the window using the getstr(3XC) set of functions. When a newline is received, the line is processed by scanw() which places the result in the appropriate <i>arg</i> s.</p> <p>With scanw() and wscanw(), the characters are read from <code>stdscr</code> and <i>win</i>, respectively. The mvscanw() and mvwscanw() functions position the cursor in the window and then call scanw().</p> |

The **vwscanw()** and **vw_scanw()** functions are similar to **wscanw()** but use a pointer to a variable parameter list as defined by either `<varargs.h>` or `<stdarg.h>`. Each application must include the appropriate header.

RETURN VALUES

On success, these functions return `OK`. Otherwise, they return `ERR`.

ERRORS

None

SEE ALSO

`getnstr(3XC)`, `mvprintw(3XC)`, `scanf(3S)`

| | | | | | | | |
|----------------------|--|------------|-------------------------------------|----------|---|----------|--|
| NAME | mvwin – move window | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int mvwin(WINDOW *win, int y, int x);</pre> | | | | | | |
| PARAMETERS | <table><tr><td>win</td><td>Is a pointer to the window to move.</td></tr><tr><td>y</td><td>Is the y (row) coordinate of the upper left corner of the window.</td></tr><tr><td>x</td><td>Is the x (column) coordinate of the upper left corner of the window.</td></tr></table> | win | Is a pointer to the window to move. | y | Is the y (row) coordinate of the upper left corner of the window. | x | Is the x (column) coordinate of the upper left corner of the window. |
| win | Is a pointer to the window to move. | | | | | | |
| y | Is the y (row) coordinate of the upper left corner of the window. | | | | | | |
| x | Is the x (column) coordinate of the upper left corner of the window. | | | | | | |
| DESCRIPTION | The mvwin() function moves the specified window (or subwindow), placing its upper left corner at the positions specified by <i>x</i> and <i>y</i> . The entire window must fit within the physical boundaries of the screen or an error results. In the case of a subwindow, the window must remain within the boundaries of the parent window. | | | | | | |
| RETURN VALUES | On success, the mvwin() function returns OK. Otherwise, it returns ERR. | | | | | | |
| ERRORS | None. | | | | | | |
| SEE ALSO | derwin(3XC) | | | | | | |

| NAME | nanosleep – high resolution sleep | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <time.h> int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);</pre> | | | | |
| DESCRIPTION | <p>The nanosleep() function causes the current thread to be suspended from execution until either the time interval specified by the <i>rqtp</i> argument has elapsed or a signal is delivered to the calling thread and its action is to invoke a signal-catching function or to terminate the process. The suspension time may be longer than requested because the argument value is rounded up to an integer multiple of the sleep resolution or because of the scheduling of other activity by the system. But, except for the case of being interrupted by a signal, the suspension time will not be less than the time specified by <i>rqtp</i>, as measured by the system clock, <code>CLOCK_REALTIME</code>.</p> <p>The use of the nanosleep() function has no effect on the action or blockage of any signal.</p> | | | | |
| RETURN VALUES | <p>If the nanosleep() function returns because the requested time has elapsed, its return value is 0.</p> <p>If the nanosleep() function returns because it has been interrupted by a signal, the function returns a value of <code>-1</code> and sets <code>errno</code> to indicate the interruption. If the <i>rmtp</i> argument is non-NULL, the <code>timespec</code> structure referenced by it is updated to contain the amount of time remaining in the interval (the requested time minus the time actually slept). If the <i>rmtp</i> argument is NULL, the remaining time is not returned.</p> <p>If nanosleep() fails, it returns <code>-1</code> and sets <code>errno</code> to indicate the error.</p> | | | | |
| ERRORS | <p>The nanosleep() function will fail if:</p> <p>EINTR The nanosleep() function was interrupted by a signal.</p> <p>EINVAL The <i>rqtp</i> argument specified a nanosecond value less than zero or greater than or equal to 1000 million.</p> <p>ENOSYS The nanosleep() function is not supported by this implementation.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |

SEE ALSO `sleep(3C)`, `attributes(5)`, `time(5)`

| | |
|----------------------|---|
| NAME | napms – sleep process for a specified length of time |
| SYNOPSIS | <pre>#include <curses.h> int napms(int ms);</pre> |
| PARAMETERS | ms Is the number of milliseconds to sleep. |
| DESCRIPTION | The napms() function sleeps for at least <i>ms</i> milliseconds. |
| RETURN VALUES | The napms() function always returns OK. |
| ERRORS | None. |
| SEE ALSO | delay_output(3XC) |

NAME netdir, netdir_getbyname, netdir_getbyaddr, netdir_free, netdir_options, taddr2uaddr, uaddr2taddr, netdir_perror, netdir_sperror, netdir_mergeaddr – generic transport name-to-address translation

SYNOPSIS

```
#include <netdir.h>
```

```
int netdir_getbyname(const struct netconfig * config, const struct nd_hostserv *
service, struct nd_addrlist ** addrs);
```

```
int netdir_getbyaddr(const struct netconfig * config, struct nd_hostservlist ** service,
const struct netbuf * netaddr);
```

```
void netdir_free(void * ptr, const int struct_type);
```

```
int netdir_options(const struct netconfig * config, const int option, const int fildes, char
* point_to_args);
```

```
char * taddr2uaddr(const struct netconfig * config, const struct netbuf * addr);
```

```
struct netbuf * uaddr2taddr(const struct netconfig * config, const char * uaddr);
```

```
void netdir_perror(char * s);
```

```
char * netdir_sperror(void);
```

DESCRIPTION

These routines provide a generic interface for name-to-address mapping that will work with all transport protocols. This interface provides a generic way for programs to convert transport specific addresses into common structures and back again. The `netconfig` structure, described on the `netconfig(4)` manual page, identifies the transport.

The `netdir_getbyname()` routine maps the machine name and service name in the `nd_hostserv` structure to a collection of addresses of the type understood by the transport identified in the `netconfig` structure. This routine returns all addresses that are valid for that transport in the `nd_addrlist` structure. The `nd_hostserv` structure contains the following members:

```
char          /* host name */
*h_serv;     /* service name */
```

The `nd_addrlist` structure contains the following members:

```
int n_cnt;          /* number of addresses */
struct netbuf *n_addrs;
```

netdir_getbyname() accepts some special-case host names. The host names are defined in `<netdir.h>`. The currently defined host names are:

| | |
|-------------------|---|
| HOST_SELF | Represents the address to which local programs will bind their endpoints. HOST_SELF differs from the host name provided by gethostname(3C) , which represents the address to which <i>remote</i> programs will bind their endpoints. |
| HOST_ANY | Represents any host accessible by this transport provider. HOST_ANY allows applications to specify a required service without specifying a particular host name. |
| HOST_SELF_CONNECT | Represents the host address that can be used to connect to the local host. |
| HOST_BROADCAST | Represents the address for all hosts accessible by this transport provider. Network requests to this address will be received by all machines. |

All fields of the `nd_hostserv` structure must be initialized.

To find the address of a given host and service on all available transports, call the **netdir_getbyname()** routine with each `struct netconfig` structure returned by **getnetconfig(3N)**.

The **netdir_getbyaddr()** routine maps addresses to service names. This routine returns *service*, a list of host and service pairs that would yield this address. If more than one tuple of host and service name is returned, then the first tuple contains the preferred host and service names:

```
struct nd_hostservlist {
    int *h_cnt; /* number of hostservs found */
    struct hostserv *h_hostservs;
}
```

The **netdir_free()** structure is used to free the structures allocated by the name to address translation routines. *ptr* points to the structure that has to be freed. The `struct_type` identifies the structure:

```

s
struct netbuf          ND_ADDR
struct nd_addrlist    ND_ADDRLIST
struct hostserv        ND_HOSTSERV
struct nd_hostservlist ND_HOSTSERVLIST
```

The universal address returned by **taddr2uaddr()** should be freed by `free()`.

The **netdir_options()** routine is used to do all transport-specific setups and option management. *fildev* is the associated file descriptor. *option*, *fildev*, and *pointer_to_args* are passed to the **netdir_options()** routine for the transport specified in *config*. Currently four values are defined for *option*:

```
ND_SET_BROADCAST
ND_SET_RESERVEDPORT
ND_CHECK_RESERVEDPORT
ND_MERGEADDR
```

The **taddr2uaddr()** and **uaddr2taddr()** routines support translation between universal addresses and TLI type `netbufs`. The **taddr2uaddr()** routine takes a `struct netbuf` data structure and returns a pointer to a string that contains the universal address. It returns `NULL` if the conversion is not possible. This is not a fatal condition as some transports may not suppose a universal address form.

uaddr2taddr() is the reverse of **taddr2uaddr()**. It returns the `struct netbuf` data structure for the given universal address.

If a transport provider does not support an option, `netdir_options` returns `-1` and the error message can be printed through **netdir_perror()** or **netdir_sperror()**.

The specific actions of each option follow.

```
ND_SET_BROADCAST
```

Sets the transport provider up to allow broadcast, if the transport supports broadcast. *fildev* is a file descriptor into the transport (i.e., the result of a `t_open` of `/dev/udp`). *pointer_to_args* is not used. If this completes, broadcast operations may be performed on file descriptor *fildev*.

```
ND_SET_RESERVEDPORT
```

Allows the application to bind to a reserved port, if that concept exists for the transport provider. *fildev* is an unbound file descriptor into the transport. If *pointer_to_args* is `NULL`, *fildev* will be bound to a reserved port. If *pointer_to_args* is a pointer to a `netbuf` structure, an attempt will be made to bind to any reserved port on the specified address.

ND_CHECK_RESERVEDPORT

Used to verify that the address corresponds to a reserved port, if that concept exists for the transport provider. *filde*s is not used. *pointer_to_args* is a pointer to a `netbuf` structure that contains the address. This option returns 0 only if the address specified in *pointer_to_args* is reserved.

ND_MERGEADDR

USED TO TAKE A "LOCAL ADDRESS" (LIKE THE 0.0.0.0 ADDRESS THAT TCP USES) AND RETURN A "REAL ADDRESS" THAT CLIENT MACHINES CAN CONNECT TO. *FILDES* IS NOT USED.

POINTER_TO_ARGS IS A POINTER TO A STRUCT ND_MERGEARG, WHICH HAS THE FOLLOWING MEMBERS:

```
char
s_uaddr;
/* server's universal address */

char
c_uaddr;
/* client's universal address */

char
m_uaddr;
/* the result */
```

If *s_uaddr* is something like 0.0.0.0.1.12, and, if the call is successful, *m_uaddr* will be set to something like 192.11.109.89.1.12. For most transports, *m_uaddr* is exactly what *s_uaddr* is.

RETURN VALUES

The `netdir_perror()` routine prints an error message on the standard output stating why one of the name-to-address mapping routines failed. The error message is preceded by the string given as an argument.

The `netdir_sperror()` routine returns a string containing an error message stating why one of the name-to-address mapping routines failed.

`netdir_sperror()` returns a pointer to a buffer which contains the error message string. This buffer is overwritten on each call. In multithreaded applications, this buffer is implemented as thread-specific data.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`gethostname(3C)`, `getnetconfig(3N)`, `getnetpath(3N)`,
`netconfig(4)`, `attributes(5)`

| | | | | | | | | | | | | | | | | | | | | | |
|-----------------------|---|----------------------|--|---------------------|--|-------------------|-------------------------------------|-----------------------|--|-----------------------|--|-----------------------|---|-----------------------|--|-----------------------|--|-----------------------|---|--------------------|---|
| NAME | newpad, pnoutrefresh, prefresh, subpad – create or refresh a pad or subpad | | | | | | | | | | | | | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> WINDOW * newpad(int <i>nlines</i>, int <i>ncols</i>); int pnoutrefresh(WINDOW * <i>pad</i>, int <i>pminrow</i>, int <i>pmincol</i>, int <i>sminrow</i>, int <i>smincol</i>, int <i>smaxrow</i>, int <i>smaxcol</i>); int prefresh(WINDOW * <i>pad</i>, int <i>pminrow</i>, int <i>pmincol</i>, int <i>sminrow</i>, int <i>smincol</i>, int <i>smaxrow</i>, int <i>smaxcol</i>); WINDOW * subpad(WINDOW * <i>orig</i>, int <i>nlines</i>, int <i>ncols</i>);</pre> | | | | | | | | | | | | | | | | | | | | |
| PARAMETERS | <table border="0" style="width: 100%;"> <tr> <td style="padding-right: 20px;"><i>nlines</i></td> <td>Is the number of lines in the pad to be created.</td> </tr> <tr> <td><i>ncols</i></td> <td>Is the number of columns in the pad to be created.</td> </tr> <tr> <td><i>pad</i></td> <td>Is a pointer to the pad to refresh.</td> </tr> <tr> <td><i>pminrow</i></td> <td>Is the row coordinate of the upper left corner of the pad rectangle to be copied</td> </tr> <tr> <td><i>pmincol</i></td> <td>Is the column coordinate of the upper left corner of the pad rectangle to be copied.</td> </tr> <tr> <td><i>sminrow</i></td> <td>Is the row coordinate of the upper left corner of the rectangle on the physical screen where pad is to be positioned.</td> </tr> <tr> <td><i>smincol</i></td> <td>Is the column coordinate of the upper left corner of the rectangle on the physical screen where pad is to be positioned.</td> </tr> <tr> <td><i>smaxrow</i></td> <td>Is the row coordinate of the lower right corner of the rectangle on the physical screen where the pad is to be positioned.</td> </tr> <tr> <td><i>smaxcol</i></td> <td>Is the column coordinate of the lower right corner of the rectangle on the physical screen where the pad is to be positioned.</td> </tr> <tr> <td><i>orig</i></td> <td>Is a pointer to the parent pad within which a sub-pad is created.</td> </tr> </table> | <i>nlines</i> | Is the number of lines in the pad to be created. | <i>ncols</i> | Is the number of columns in the pad to be created. | <i>pad</i> | Is a pointer to the pad to refresh. | <i>pminrow</i> | Is the row coordinate of the upper left corner of the pad rectangle to be copied | <i>pmincol</i> | Is the column coordinate of the upper left corner of the pad rectangle to be copied. | <i>sminrow</i> | Is the row coordinate of the upper left corner of the rectangle on the physical screen where pad is to be positioned. | <i>smincol</i> | Is the column coordinate of the upper left corner of the rectangle on the physical screen where pad is to be positioned. | <i>smaxrow</i> | Is the row coordinate of the lower right corner of the rectangle on the physical screen where the pad is to be positioned. | <i>smaxcol</i> | Is the column coordinate of the lower right corner of the rectangle on the physical screen where the pad is to be positioned. | <i>orig</i> | Is a pointer to the parent pad within which a sub-pad is created. |
| <i>nlines</i> | Is the number of lines in the pad to be created. | | | | | | | | | | | | | | | | | | | | |
| <i>ncols</i> | Is the number of columns in the pad to be created. | | | | | | | | | | | | | | | | | | | | |
| <i>pad</i> | Is a pointer to the pad to refresh. | | | | | | | | | | | | | | | | | | | | |
| <i>pminrow</i> | Is the row coordinate of the upper left corner of the pad rectangle to be copied | | | | | | | | | | | | | | | | | | | | |
| <i>pmincol</i> | Is the column coordinate of the upper left corner of the pad rectangle to be copied. | | | | | | | | | | | | | | | | | | | | |
| <i>sminrow</i> | Is the row coordinate of the upper left corner of the rectangle on the physical screen where pad is to be positioned. | | | | | | | | | | | | | | | | | | | | |
| <i>smincol</i> | Is the column coordinate of the upper left corner of the rectangle on the physical screen where pad is to be positioned. | | | | | | | | | | | | | | | | | | | | |
| <i>smaxrow</i> | Is the row coordinate of the lower right corner of the rectangle on the physical screen where the pad is to be positioned. | | | | | | | | | | | | | | | | | | | | |
| <i>smaxcol</i> | Is the column coordinate of the lower right corner of the rectangle on the physical screen where the pad is to be positioned. | | | | | | | | | | | | | | | | | | | | |
| <i>orig</i> | Is a pointer to the parent pad within which a sub-pad is created. | | | | | | | | | | | | | | | | | | | | |

DESCRIPTION

The **newpad()** function creates a new pad with the specified number of lines and columns. A pointer to the new pad structure is returned. A pad differs from a window in that it is not restricted to the size of the physical screen. It is useful when only part of a large window will be displayed at any one time.

Automatic refreshes by scrolling or echoing of input do not take place when pads are used. Pads have their own refresh commands, **prefresh()** and **pnoutrefresh()** .

The **prefresh()** function copies the specified portion of the logical pad to the terminal screen. The parameters *pmincol* and *pminrow* specify the upper left corner of the rectangular area of the pad to be displayed. The lower right coordinate of the rectangular area of the pad that is to be displayed is calculated from the screen parameters (*sminrow* , *smincol* , *smaxrow* , *smaxcol*).

This function calls the **pnoutrefresh()** function to copy the specified portion of *pad* to the terminal screen and the **doupdate(3XC)** function to do the actual update. The logical cursor is copied to the same location in the physical window unless **leaveok(3XC)** is enabled (in which case, the cursor is placed in a position that the program finds convenient).

When outputting several pads at once, it is often more efficient to call the **pnoutrefresh()** and **doupdate()** functions directly. A call to **pnoutrefresh()** for each pad first, followed by only one call to **doupdate()** to update the screen, results in one burst of output, fewer characters sent, and less CPU time used.

The **subpad()** function creates a sub-pad within the pad *orig* with the specified number of lines and columns. A pointer to the new pad structure is returned. The sub-pad is positioned in the middle of *orig* . Any changes made to one pad affect the other. **touchwin(3XC)** or **touchline(3XC)** will likely have to be called on pad *orig* to correctly update the window.

RETURN VALUES

On success, the **newpad()** and **subpad()** functions returns a pointer to the new pad data structure. Otherwise, they return a null pointer.

On success, the **pnoutrefresh()** and **prefresh()** functions return **OK** . Otherwise, they return **ERR** .

SEE ALSO

clearok(3XC) , **doupdate(3XC)** , **is_linetouched(3XC)** , **pechochar(3XC)**

| NAME | nextafter – next representable double-precision floating-point number | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | cc [<i>flag ...</i>] <i>file ...</i> -lm [<i>library ...</i>] #include <math.h> double nextafter (double <i>x</i> , double <i>y</i>); | | | | |
| DESCRIPTION | The nextafter() function computes the next representable double-precision floating-point value following <i>x</i> in the direction of <i>y</i> . Thus, if <i>y</i> is less than <i>x</i> , nextafter() returns the largest representable floating-point number less than <i>x</i> . | | | | |
| RETURN VALUES | The nextafter() function returns the next representable double-precision floating-point value following <i>x</i> in the direction of <i>y</i> . If <i>x</i> or <i>y</i> is NaN, then nextafter() returns NaN. If <i>x</i> is finite and the correct function value would overflow, nextafter() returns ±HUGE_VAL (according to the sign of <i>x</i>) and sets <i>errno</i> to ERANGE. | | | | |
| ERRORS | The nextafter() function will fail if: ERANGE the correct value would overflow. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | attributes(5) , | | | | |

| | |
|----------------------|---|
| NAME | nice – change priority of a process |
| SYNOPSIS | <pre>/usr/ucb/cc[<i>flag ...</i>] <i>file ...</i> #include<unistd.h> int nice(<i>incr</i>); int <i>incr</i>;</pre> |
| DESCRIPTION | <p>The scheduling priority of the process is augmented by <i>incr</i>. Positive priorities get less service than normal. Priority 10 is recommended to users who wish to execute long-running programs without undue impact on system performance.</p> <p>Negative increments are illegal, except when specified by the privileged user. The priority is limited to the range –20 (most urgent) to 20 (least). Requests for values above or below these limits result in the scheduling priority being set to the corresponding limit.</p> <p>The priority of a process is passed to a child process by <code>fork(2)</code>. For a privileged process to return to normal priority from an unknown state, <code>nice()</code> should be called successively with arguments –40 (goes to priority –20 because of truncation), 20 (to get to 0), then 0 (to maintain compatibility with previous versions of this call).</p> |
| RETURN VALUES | Upon successful completion, <code>nice()</code> returns 0. Otherwise, a value of –1 is returned and <code>errno</code> is set to indicate the error. |
| ERRORS | The priority is not changed if: EPERM The value of <i>incr</i> specified was negative, and the effective user ID is not the privileged user. |
| SEE ALSO | <code>nice(1)</code> , <code>renice(1)</code> , <code>fork(2)</code> , <code>pricntl(2)</code> , <code>getpriority(3C)</code> |
| NOTES | Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-threaded applications is unsupported. |

NAME nis_db, db_initialize, db_create_table, db_destroy_table, db_first_entry, db_next_entry, db_reset_next_entry, db_list_entries, db_remove_entry, db_add_entry, db_table_exists, db_unload_table, db_checkpoint, db_standby, db_free_result – NIS+ Database access functions

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lnisdb

-lnsl
[
library
... ]
#include <rpcsvc/nis.h>
#include <rpcsvc/nis_db.h>

bool db_initialize(char * dictionary_pathname);
db_status db_create_table(char * table_name, table_obj * table);
db_status db_destroy_table(* table_name);
db_result * db_first_entry(char * table_name, int numattrs, nis_attr * attrs);
db_result * db_next_entry(char * table_name, db_next_desc * next_handle);
db_result * db_reset_next_entry(char * table_name, db_next_desc * next_handle);
db_result * db_list_entries(char * table_name, int numattrs, nis_attr * attrs);
db_result * db_remove_entry(char * table_name, int numattrs, nis_attr * attrs);
db_result * db_add_entry(char * table_name, int numattrs, nis_attr * attrs, entry_obj *
entry);
db_status db_table_exists(char * table_name);
db_status db_unload_table(char * table_name);
db_status db_checkpoint(char * table_name);
db_status db_standby(char * table_name);
void db_free_result(db_result *);
```

DESCRIPTION

These functions describe the interface between the NIS+ server and the underlying database. They are defined in the shared library `/usr/lib/libnisdb.so`.

The interface is a simple subset of a complete relational database and provides just those items that are needed by the NIS+ server daemon. When you replace the database, your interface routines should match these exactly. Also note that the database is responsible for verifying that the objects passed do not exceed the internal limits of the database being used.

The database's performance will directly affect the performance of the server. The default information base that is provided with NIS+ is the Structured Storage Manager (SSM). This is a memory based database that has been tuned for NIS+.

These routines should not be invoked by any NIS+ client. NIS+ clients should use the NIS+ tables API described in `nis_tables(3N)`.

These routines only use the `table_obj`, `entry_obj` and the `nis_attr` structures defined in `<rpcsvc/nis.h>`. The NIS+ directory is itself stored in a table by the service daemon. This table has two columns, one searchable with the name of the object in it, the other non-searchable with binary XDRed data in it. The NIS+ server converts directory lookup requests in the namespace into table searches. The table it searches in response to these requests will have the same name as the directory of the name it is searching for.

The structure returned by the DB access routines is defined as:

```
enum db_status {DB_SUCCESS, DB_NOTFOUND, DB_NOTUNIQUE, DB_BADTABLE,
                DB_BADQUERY, DB_BADOBJECT, DB_MEMORY_LIMIT, DB_STORAGE_LIMIT,
                DB_INTERNAL_ERROR };
struct db_result {
\011    db_status\011    status;\011\011    /* Result status */
\011    db_next_desc\011nextinfo;\011\011    /* descriptor */
\011    struct {
\011\011        int_t\011        objects_len;
\011\011        entry_obj\011*objects_val;
\011} objects;\011\011\011\011    /* A variable list
\011\011\011\011\011\011        of objects */
\011long\011ticks;\011\011\011\011    /* execution time in
\011\011\011\011\011\011        microseconds */
};
```

For a complete description of NIS+ objects, see `nis_objects(3N)`.

The structure `db_next_desc` should be used as an opaque handle for `db_next_entry()` and `db_reset_next_entry()`.

The `nis_attr` structure used in `db_first_entry` and other related functions is defined as follows:

```
struct nis_attr {
\011     char\011*zattr_ndx;
\011     struct {
\011\011         uint_t\011zattr_val_len;
\011\011         char\011*zattr_val_val;
\011     } zattr_val;
};
```

`zattr_ndx` is the name of the attribute. `zattr_val_len` is the value of the attribute `zattr_val_val`.

In `db_result`, the *objects* array contains objects if and only if the result returned in the *status* variable is `DB_SUCCESS`. A null pointer, instead of a pointer to a `db_result` structure, is returned if there is insufficient memory to create the structure.

db_initialize() is called prior to any interaction with the database. It takes as argument the pathname of the file that contains, or will contain, catalog information associated with the database.

db_create_table() creates a new table using the given table name and the table object. It returns `TRUE` if the table was successfully created; `FALSE` otherwise.

db_destroy_table() destroys the table of the given name. It returns `TRUE` if the destruction was successful; `FALSE` otherwise.

db_first_entry() returns a copy of the first entry in the specified table that satisfies the given attributes. If no attributes are supplied, a copy of the first entry in the table is returned. *attrs* is an array of `nis_attr` structure with *numattrs* number of elements. The returned structure, `db_result`, contains a structure, `db_next_desc`, to be used as an argument to **db_next_entry()** or **db_reset_next_entry()**. `db_next_desc` should only be used only as an opaque handle. **db_free_result()** can be used to free up the returned `db_result` structure.

db_next_entry() returns a copy of the next entry as indicated by the *next_handle*. An initial call to **db_first_entry()**, followed by a sequence of calls to **db_next_entry()**, can be used to successfully obtain entries of an entire table or entries that satisfy the attributes supplied to **db_first_entry()**. **db_free_result()** can be used to free up the returned `db_result` structure.

db_reset_next_entry() terminates the **db_first_entry()** / **db_next_entry()** sequence as indicated by *next_handle*, freeing any resources that have been used to maintain the sequence. After a call to **db_reset_next_entry()**, a call to

db_next_entry() using the same *next_handle* would fail, returning a `DB_BADQUERY` reply. **db_free_result()** can be used to free up the returned `db_result` structure.

db_list_entries() returns copies of entries that satisfy the given attributes. **db_free_result()** can be used to free up the returned `db_result` structure. `attrs` is an array of `nis_attr` structure with *numattrs* number of elements.

db_remove_entry() removes all entries that satisfy the given attributes. **db_free_result()** can be used to free up the returned `db_result` structure. `attrs` is an array of `nis_attr` structure with *numattrs* number of elements.

db_add_entry() adds a copy of the given object to the specified table, replacing the one identified by the given attributes. If the given attributes identify more than one object, `DB_NOTUNIQUE` is returned. If no object is identified by the given attributes, the object is added. `attrs` is an array of `nis_attr` structure with *numattrs* number of elements. **db_free_result()** can be used to free up the returned `db_result` structure.

db_table_exists() provides an efficient way for the NIS+ service to detect that a table exists. This increases response time to the client and lowers the load on the server.

db_unload_table() is used by the service to unload or deactivate tables that are not currently being used. The service internally keeps track of access patterns to tables and will unload those tables that have not been accessed for a while. By unloading infrequently accessed tables, the service can minimize the amount of system resources for efficient operation.

db_checkpoint() organizes the contents of the table in a more efficient manner. Checkpointing may mean different things to different types of databases. It does not affect the logical contents of the table — operations and queries should return the same result before and after a checkpoint. For example, in a log-based system, checkpointing may mean incorporating log entries of updates accumulated since the previous checkpoint into the table.

db_free_result() frees up the space allocated by various functions listed on this manual page that return a `db_result` structure.

db_standby() is an advisory call to the database manager. This call informs the database that activity has slowed down and it can free up unnecessary resources such as file descriptors.

PROGRAMMING

Most of the routines in this library use an NIS+ *name* to identify the object that the user desires. The name must be in canonical form before being passed to the database because one server may be serving several namespaces and discrimination of the requested objects is accomplished by comparing the domain names.

DIAGNOSTICS

| | |
|-------------------|---|
| DB_SUCCESS | The query or operation completed successfully and returned status. |
| DB_NOTFOUND | The name or entry that was named in the argument did not exist. |
| DB_NOTUNIQUE | An attempt was made to remove an entry from a table that is not uniquely specified. |
| DB_BADQUERY | The query that was submitted to the database was invalid (for example, it might name some nonexistent fields). |
| DB_BADTABLE | The table was corrupted. |
| DB_BADOBJECT | The fields of the object does not conform to the fields of the table to which it is being added. |
| DB_MEMORY_LIMIT | There is insufficient memory to complete the operation requested. |
| DB_STORAGE_LIMIT | There is insufficient file storage available to complete the operation requested. |
| DB_INTERNAL_ERROR | An internal error was encountered during the execution of the operation requested (either a programming error or an unrecoverable exception). |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

rpc.nisd(1M) , **nis_objects(3N)** , **nisfiles(4)** , **attributes(5)**

| | |
|--------------------|--|
| NAME | nis_error, nis_sperrno, nis_perror, nis_lerror, nis_sperror, nis_sperror_r – display NIS+ error messages |
| SYNOPSIS | <pre>cc [flag ...] file ... -lnsl [library ...] #include <rpcsvc/nis.h> char * nis_sperrno(nis_error status); void nis_perror(nis_error status, char * label); void nis_lerror(nis_error status, char * label); char * nis_sperror_r(nis_error status, char * label, char * buf, int length); char * nis_sperror(nis_error status, char * label);</pre> |
| DESCRIPTION | <p>These functions convert NIS+ status values into text strings.</p> <p>nis_sperrno() simply returns a pointer to a string constant which is the error string.</p> <p>nis_perror() prints the error message corresponding to <i>status</i> as “<i>label</i> : error message ” on standard error.</p> <p>nis_lerror() sends the error text to syslog(3) at level LOG_ERR.</p> <p>The function nis_sperror_r() , returns a pointer to a string that can be used or copied using the strdup() function (See string(3C)). The caller must supply a string buffer, <i>buf</i> , large enough to hold the error string (a buffer size of 128 bytes is guaranteed to be sufficiently large). <i>status</i> and <i>label</i> are the same as for nis_perror() . The pointer returned by nis_sperror_r() is the same as <i>buf</i> , that is, the pointer returned by the function is a pointer to <i>buf</i> . <i>length</i> specifies the number of characters to copy from the error string to <i>buf</i> .</p> <p>The last function, nis_sperror() , is similar to nis_sperror_r() except that the string is returned as a pointer to a buffer that is reused on each call. nis_sperror_r() is the preferred interface, since it is suitable for single-threaded and multi-threaded programs.</p> |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

niserror(1) , **string(3C)** , **syslog(3)** , **attributes(5)**

NOTES

When compiling multithreaded applications, see **Intro(3)** , *Notes On Multithread Applications* , for information about the use of the **_REENTRANT** flag.

| | |
|--------------------|--|
| NAME | nis_groups, nis_ismember, nis_addmember, nis_removemember, nis_creategroup, nis_destroygroup, nis_verifygroup, nis_print_group_entry – NIS+ group manipulation functions |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] #include <rpcsvc/nis.h> bool_t nis_ismember(nis_name <i>principal</i>, nis_name <i>group</i>); nis_error nis_addmember(nis_name <i>member</i>, nis_name <i>group</i>); nis_error nis_removemember(nis_name <i>member</i>, nis_name <i>group</i>); nis_error nis_creategroup(nis_name <i>group</i>, uint_t <i>flags</i>); nis_error nis_destroygroup(nis_name <i>group</i>); void nis_print_group_entry(nis_name <i>group</i>); nis_error nis_verifygroup(nis_name <i>group</i>);</pre> |
| DESCRIPTION | <p>These functions manipulate NIS+ groups. They are used by NIS+ clients and servers, and are the interfaces to the group authorization object.</p> <p>The names of NIS+ groups are syntactically similar to names of NIS+ objects but they occupy a separate namespace. A group named "a.b.c.d." is represented by a NIS+ group object named "a.groups_dir.b.c.d."; the functions described here all expect the name of the group, not the name of the corresponding group object.</p> <p>There are three types of group members:</p> <ul style="list-style-type: none"> ■ An <i>explicit</i> member is just a NIS+ principal-name, for example "wickedwitch.west.oz." ■ An <i>implicit</i> ("domain") member, written "*.west.oz.", means that all principals in the given domain belong to this member. No other forms of wildcarding are allowed: "wickedwitch.*.oz." is invalid, as is "wickedwitch.west.*.". Note that principals in subdomains of the given domain are <i>not</i> included. |

- A *recursive* ("group") member, written "@cowards.oz.", refers to another group; all principals that belong to that group are considered to belong here.

Any member may be made *negative* by prefixing it with a minus sign ('-'). A group may thus contain explicit, implicit, recursive, negative explicit, negative implicit, and negative recursive members.

A principal is considered to belong to a group if it belongs to at least one non-negative group member of the group and belongs to no negative group members.

The **nis_ismember()** function returns TRUE if it can establish that *principal* belongs to *group* ; otherwise it returns FALSE.

The **nis_addmember()** and **nis_removemember()** functions add or remove a member. They do not check whether the member is valid. The user must have read and modify rights for the group in question.

The **nis_creategroup()** and **nis_destroygroup()** functions create and destroy group objects. The user must have create or destroy rights, respectively, for the *groups_dir* directory in the appropriate domain. The parameter *flags* to **nis_creategroup()** is currently unused and should be set to zero.

The **nis_print_group_entry()** function lists a group's members on the standard output.

The **nis_verifygroup()** function returns NIS_SUCCESS if the given group exists, otherwise it returns an error code.

EXAMPLES

EXAMPLE 1 Simple Memberships

Given a group `sadsouls.oz.` with members `tinman.oz.`, `lion.oz.`, and `scarecrow.oz.`, the function call

```
bool_var = nis_ismember("lion.oz.", "sadsouls.oz.");
```

will return 1 (TRUE) and the function call

```
bool_var = nis_ismember("toto.oz.", "sadsouls.oz.");
```

will return 0 (FALSE).

EXAMPLE 2 Implicit Memberships

Given a group `baddies.oz.`, with members `wickedwitch.west.oz.` and `*.monkeys.west.oz.`, the function call

```
bool_var = nis_ismember("hogan.monkeys.west.oz.", "baddies.oz.");
```

will return 1 (TRUE) because any principal from the `monkeys.west.oz.` domain belongs to the implicit group `*.monkeys.west.oz.`, but the function call

```
bool_var = nis_ismember("hogan.big.monkeys.west.oz.", "baddies.oz.");
```

will return 0 (FALSE).

EXAMPLE 3 Recursive Memberships

Given a group `goodandbad.oz.`, with members `toto.kansas`, `@sadsouls.oz.`, and `@baddies.oz.`, and the groups `sadsouls.oz.` and `baddies.oz.` defined above, the function call

```
bool_var = nis_ismember("wickedwitch.west.oz.", "goodandbad.oz.");
```

will return 1 (TRUE), because `wickedwitch.west.oz.` is a member of the `baddies.oz.` group which is recursively included in the `goodandbad.oz.` group.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`nisgrpadm(1)`, `nis_objects(3N)`, `attributes(5)`

NOTES

These functions only accept fully-qualified NIS+ names.

A group is represented by a NIS+ object (see `nis_objects(3N)`) with a variant part that is defined in the `group_obj` structure. It contains the following fields:

```
uint_t\011gr_flags;\011/* Interpretation Flags
\011\011\011(currently unused) */
struct {
\011uint_t\011gr_members_len;
\011nis_name\011*gr_members_val;
} gr_members;\011\011/* Array of members */
```

NIS+ servers and clients maintain a local cache of expanded groups to enhance their performance when checking for group membership. Should the membership of a group change, servers and clients with that group cached will not see the change until either the group cache has expired or it is explicitly flushed. A server's cache may be flushed programmatically by calling the `nis_servstate()` function with tag `TAG_GCACHE` and a value of 1.

There are currently no known methods for `nis_ismember()`, `nis_print_group_entry()`, and `nis_verifygroup()` to get their answers from only the master server.

| | |
|--------------------|---|
| NAME | nis_local_names, nis_local_directory, nis_local_host, nis_local_group, nis_local_principal - NIS+ local names |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] #include <rpcsvc/nis.h> nis_name nis_local_directory(void); nis_name nis_local_host(void); nis_name nis_local_group(void); nis_name nis_local_principal(void);</pre> |
| DESCRIPTION | <p>These functions return several default NIS+ names associated with the current process.</p> <p>nis_local_directory() returns the name of the NIS+ domain for this machine. This is currently the same as the Secure RPC domain returned by the sysinfo(2) system call.</p> <p>nis_local_host() returns the NIS+ name of the current machine. This is the fully qualified name for the host and is either the value returned by the gethostname(3C) function or, if the host name is only partially qualified, the concatenation of that value and the name of the NIS+ directory. Note that if a machine's name and address cannot be found in the local NIS+ directory, its hostname must be fully qualified.</p> <p>nis_local_group() returns the name of the current NIS+ group name. This is currently set by setting the environment variable NIS_GROUP to the groupname.</p> <p>nis_local_principal() returns the NIS+ principal name for the user associated with the effective UID of the calling process. This function maps the effective uid into a principal name by looking for a LOCAL type credential (see nisaddcred(1M)) in the table named <i>cred.org_dir</i> in the default domain.</p> |

Note: The result returned by these routines is a pointer to a data structure with the NIS+ library, and should be considered a “read-only” result and should not be modified.

ENVIRONMENT VARIABLES

NIS_GROUP This variable contains the name of the local NIS+ group. If the name is not fully qualified, the value returned by **nis_local_directory()** will be concatenated to it.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

nisdefaults(1), **nisaddcred(1M)**, **sysinfo(2)**, **gethostname(3C)**, **nis_names(3N)**, **nis_objects(3N)**, **attributes(5)**

| | |
|--------------------|--|
| NAME | nis_names, nis_lookup, nis_add, nis_remove, nis_modify, nis_freeresult – NIS+ namespace functions |
| SYNOPSIS | <pre>cc [flag ...] file ... -lns1 [library ...] #include <rpcsvc/nis.h> nis_result * nis_lookup(nis_name name, uint_t flags); nis_result * nis_add(nis_name name, nis_object * obj); nis_result * nis_remove(nis_name name, nis_object * obj); nis_result * nis_modify(nis_name name, nis_object * obj); void nis_freeresult(nis_result * result);</pre> |
| DESCRIPTION | <p>These functions are used to locate and manipulate all NIS+ objects (see nis_objects(3N)) except the NIS+ entry objects. To look up the NIS+ entry objects within a NIS+ table, refer to nis_subr(3N).</p> <p>nis_lookup() resolves a NIS+ name and returns a copy of that object from a NIS+ server. nis_add() and nis_remove() add and remove objects to the NIS+ namespace, respectively. nis_modify() can change specific attributes of an object that already exists in the namespace.</p> <p>These functions should be used only with names that refer to an NIS+ Directory, NIS+ Table, NIS+ Group, or NIS+ Private object. If a name refers to an NIS+ entry object, the functions listed in nis_subr(3N) should be used.</p> <p>nis_freeresult() frees all memory associated with a <code>nis_result</code> structure. This function must be called to free the memory associated with a NIS+ result. nis_lookup(), nis_add(), nis_remove(), and nis_modify() all return a pointer to a <code>nis_result</code> structure which <i>must</i> be freed by calling nis_freeresult() when you have finished using it. If one or more of the objects returned in the structure need to be retained, they can be copied with nis_clone_object(3N) (see nis_subr(3N)).</p> <p>nis_lookup() takes two parameters, the name of the object to be resolved in <code>name</code>, and a flags parameter, <code>flags</code>, which is defined below. The object name is</p> |

expected to correspond to the syntax of a non-indexed NIS+ name (see `nis_tables(3N)`). The `nis_lookup()` function is the *only* function from this group that can use a non-fully qualified name. If the parameter *name* is not a fully qualified name, then the flag `EXPAND_NAME` *must* be specified in the call. If this flag is not specified, the function will fail with the error `NIS_BADNAME`.

The *flags* parameter is constructed by logically ORing zero or more flags from the following list.

| | |
|---------------------------|---|
| <code>FOLLOW_LINKS</code> | When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the linked object is itself a link, then this process will iterate until the either a object is found that is not a <i>LINK</i> type object, or the library has followed 16 links. |
| <code>HARD_LOOKUP</code> | When specified, the client library will retry the lookup until it is answered by a server. Using this flag will cause the library to block until at least one NIS+ server is available. If the network connectivity is impaired, this can be a relatively long time. |
| <code>NO_CACHE</code> | When specified, the client library will bypass any object caches and will get the object from either the master NIS+ server or one of its replicas. |
| <code>MASTER_ONLY</code> | When specified, the client library will bypass any object caches and any domain replicas and fetch the object from the NIS+ master server for the object’s domain. This insures that the object returned is up to date at the cost of a possible performance degradation and failure if the master server is unavailable or physically distant. |
| <code>EXPAND_NAME</code> | When specified, the client library will attempt to expand a partially qualified name by calling the function <code>nis_getnames()</code> (see <code>nis_subr(3N)</code>) which uses the environment variable <code>NIS_PATH</code> . |

The status value may be translated to ascii text using the function `nis_sperrno()` (see `nis_error(3N)`).

On return, the *objects* array in the result will contain one and possibly several objects that were resolved by the request. If the `FOLLOW_LINKS` flag was present, on success the function could return several entry objects if the link in

question pointed within a table. If an error occurred when following a link, the objects array will contain a copy of the link object itself.

The function **nis_add()** will take the object *obj* and add it to the NIS+ namespace with the name *name*. This operation will fail if the client making the request does not have the *create* access right for the domain in which this object will be added. The parameter *name* must contain a fully qualified NIS+ name. The object members *zo_name* and *zo_domain* will be constructed from this name. This operation will fail if the object already exists. This feature prevents the accidental addition of objects over another object that has been added by another process.

The function **nis_remove()** will remove the object with name *name* from the NIS+ namespace. The client making this request must have the *destroy* access right for the domain in which this object resides. If the named object is a link, the link is removed and *not* the object that it points to. If the parameter *obj* is not NULL, it is assumed to point to a copy of the object being removed. In this case, if the object on the server does not have the same object identifier as the object being passed, the operation will fail with the NIS_NOTSAMEOBJ error. This feature allows the client to insure that it is removing the desired object. The parameter *name* must contain a fully qualified NIS+ name.

The function **nis_modify()** will modify the object named by *name* to the field values in the object pointed to by *obj*. This object should contain a copy of the object from the name space that is being modified. This operation will fail with the error NIS_NOTSAMEOBJ if the object identifier of the passed object does not match that of the object being modified in the namespace.

Normally the contents of the member *zo_name* in the *nis_object* structure would be constructed from the name passed in the *name* parameter. However, if it is non-null the client library will use the name in the *zo_name* member to perform a rename operation on the object. This name *must not* contain any unquoted '.' (dot) characters. If these conditions are not met the operation will fail and return the NIS_BADNAME error code.

Results

These functions return a pointer to a structure of type `nis_result` :

```
struct nis_result {
    \011nis_error status;
    \011struct {
        \011\011uint_t\011objects_len;

        \011\011nis_object\011*objects_val;
    \011} objects;
    \011netobj\011cookie;
    \011uint32_t\011zticks;
    \011uint32_t\011dticks;
    \011uint32_t\011aticks;
    \011uint32_t\011cticks;
};
```

The *status* member contains the error status of the the operation. A text message that describes the error can be obtained by calling the function **nis_sperno()** (see **nis_error(3N)**).

The *objects* structure contains two members. *objects_val* is an array of *nis_object* structures; *objects_len* is the number of cells in the array. These objects will be freed by the call to **nis_freeresult()** . If you need to keep a copy of one or more objects, they can be copied with the function **nis_clone_object()** and freed with the function **nis_destroy_object()** (see **nis_server(3N)**). Refer to **nis_objects(3N)** for a description of the *nis_object* structure.

The various ticks contain details of where the time was taken during a request. They can be used to tune one's data organization for faster access and to compare different database implementations (see **nis_db(3N)**).

zticks The time spent in the NIS+ service itself. This count starts when the server receives the request and stops when it sends the reply.

dticks The time spent in the database backend. This time is measured from the time a database call starts, until the result is returned. If the request results in multiple calls to the database, this is the sum of all the time spent in those calls.

aticks The time spent in any "accelerators" or caches. This includes the time required to locate the server needed to resolve the request.

cticks The total time spent in the request. This clock starts when you enter the client library and stops when a result is returned. By subtracting the sum of the other ticks values from this value, you can obtain the local overhead of generating a NIS+ request.

Subtracting the value in *dticks* from the value in *zticks* will yield the time spent in the service code itself. Subtracting the sum of the values in *zticks* and *aticks* from the value in *cticks* will yield the time spent in the client library itself.

Note: all of the tick times are measured in microseconds.

RETURN VALUES

The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.

| | |
|---------------|---|
| NIS_SUCCESS | The request was successful. |
| NIS_S_SUCCESS | The request was successful, however the object returned came from an object cache and not directly from the |

| | |
|----------------------------------|---|
| | server. If you do not wish to see objects from object caches you must specify the flag <code>NO_CACHE</code> when you call the lookup function. |
| <code>NIS_NOTFOUND</code> | The named object does not exist in the namespace. |
| <code>NIS_CACHEEXPIRED</code> | The object returned came from an object cache that has <i>expired</i> . The time to live value has gone to zero and the object may have changed. If the flag <code>NO_CACHE</code> was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object. |
| <code>NIS_NAMEUNREACHABLE</code> | A server for the directory of the named object could not be reached. This can occur when there is a network partition or all servers have crashed. See the <code>HARD_LOOKUP</code> flag. |
| <code>NIS_UNKNOWNOBJ</code> | The object returned is of an unknown type. |
| <code>NIS_TRYAGAIN</code> | The server connected to was too busy to handle your request. For the <i>add</i> , <i>remove</i> , and <i>modify</i> operations this is returned when either the master server for a directory is unavailable or it is in the process of checkpointing its database. It can also be returned when the server is updating its internal state. And in the case of <code>nis_list()</code> if the client specifies a callback and the server does not have enough resources to handle the callback. |
| <code>NIS_SYSTEMERROR</code> | A generic system error occurred while attempting the request. Most commonly the server has crashed or the database has become corrupted. Check the syslog record for error messages from the server. |

| | |
|-------------------|---|
| NIS_NOT_ME | A request was made to a server that does not serve the name in question. Normally this will not occur, however if you are not using the built in location mechanism for servers you may see this if your mechanism is broken. |
| NIS_NOMEMORY | Generally a fatal result. It means that the service ran out of heap space. |
| NIS_NAMEEXISTS | An attempt was made to add a name that already exists. To add the name, first remove the existing name and then add the new object or modify the existing named object. |
| NIS_NOTMASTER | An attempt was made to update the database on a replica server. |
| NIS_INVALIDOBJ | The object pointed to by <i>obj</i> is not a valid NIS+ object. |
| NIS_BADNAME | The name passed to the function is not a legal NIS+ name. |
| NIS_LINKNAMEERROR | The name passed resolved to a <i>LINK</i> type object and the contents of the link pointed to an invalid name. |
| NIS_NOTSAMEOBJ | An attempt to remove an object from the namespace was aborted because the object that would have been removed was not the same object that was passed in the request. |
| NIS_NOSUCHNAME | This hard error indicates that the named directory of the table object does not exist. This occurs when the server that should be the parent of the server that serves the table, does not know about the directory in which the table resides. |
| NIS_NOSUCHTABLE | The named table does not exist. |

| | |
|---------------|---|
| NIS_MODFAIL | The attempted modification failed. |
| NIS_FOREIGNNS | The name could not be completely resolved. When the name passed to the function would resolve in a namespace that is outside the NIS+ name tree, this error is returned with a NIS+ object of type DIRECTORY , which contains the type of namespace and contact information for a server within that namespace. |
| NIS_RPCERROR | This fatal error indicates the RPC subsystem failed in some way. Generally there will be a <code>syslog(3)</code> message indicating why the RPC request failed. |

ENVIRONMENT VARIABLES

NIS_ PATH If the flag `EXPAND_NAME` is set, this variable is the search path used by `nis_lookup()` .

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`nis_error(3N)` , `nis_objects(3N)` , `nis_server(3N)` , `nis_subr(3N)` , `nis_tables(3N)` , `attributes(5)`

NOTES

You cannot modify the name of an object if that modification would cause the object to reside in a different domain.

You cannot modify the schema of a table object.

| | |
|--------------------------|--|
| NAME | nis_objects - NIS+ object formats |
| SYNOPSIS | cc [<i>flag ...</i>] <i>file ...</i> -lnsl [<i>library ...</i>] /usr/include/rpcsvc/nis_objects.x |
| DESCRIPTION | |
| Common Attributes | <p>The NIS+ service uses a variant record structure to hold the contents of the objects that are used by the NIS+ service. These objects all share a common structure which defines a set of attributes that all objects possess. The <code>nis_object</code> structure contains the following members:</p> <pre>typedef char *nis_name; struct nis_object { nis_oid zo_oid; nis_name zo_name; nis_name zo_owner; nis_name zo_group; nis_name zo_domain; uint_t zo_access; uint32_t zo_ttl; objdata zo_data; };</pre> <p>In this structure, the first member <code>zo_oid</code>, is a 64 bit number that uniquely identifies this instance of the object on this server. This member is filled in by the server when the object is created and changed by the server when the object is modified. When used in conjunction with the object's name and domain it uniquely identifies the object in the entire NIS+ namespace.</p> <p>The second member, <code>zo_name</code>, contains the leaf name of the object. This name is <i>never</i> terminated with a '.' (dot). When an object is created or added to the namespace, the client library will automatically fill in this field and the domain name from the name that was passed to the function.</p> <p><code>zo_domain</code> contains the name of the NIS+ domain to which this object belongs. This information is useful when tracking the parentage of an object from a cache. When used in conjunction with the members <code>zo_name</code> and <code>zo_oid</code>, it uniquely identifies an object. This makes it possible to always reconstruct the name of an object by using the code fragment</p> <pre>sprintf(buf, "%s.%s", obj->zo_name, obj->zo_domain);</pre> |

The `zo_owner` and `zo_group` members contain the NIS+ names of the object's principal owner and group owner, respectively. Both names *must be* NIS+ fully qualified names. However, neither name can be used directly to identify the object they represent. This stems from the condition that NIS+ uses itself to store information that it exports.

The `zo_owner` member contains a fully qualified NIS+ name of the form *principal.domain*. This name is called a NIS+ principal name and is used to identify authentication information in a credential table. When the server constructs a search query of the form

```
[cname=principal],cred.org_dir.domain.
```

The query will return to the server credential information about *principal* for all flavors of RPC authentication that are in use by that principal. When an RPC request is made to the server, the authentication flavor is extracted from the request and is used to find out the NIS+ principal name of the client. For example, if the client is using the AUTH_DES authentication flavor, it will include in the authentication credentials the network name or *netname* of the user making the request. This netname will be of the form

```
unix.UID@domain
```

The NIS+ server will then construct a query on the credential database of the form

```
[auth_name=netname,auth_type=AUTH_DES],cred.org_dir.domain.
```

This query will return an entry which contains a principal name in the first column. This NIS+ principal name is used to control access to NIS+ objects.

The group owner for the object is treated differently. The group owner member is optional (it should be the null string if not present) but must be fully qualified if present. A group name takes the form

```
group.domain.
```

which the server then maps into a name of the form

```
group.groups_dir.domain.
```

The purpose of this mapping is to prevent NIS+ group names from conflicting with user specified domain or table names. For example, if a domain was called *engineering.foo.com.*, then without the mapping a NIS+ group of the same name to represent members of engineering would not be possible. The contents of groups are lists of NIS+ principal names which are used exactly like the `zo_owner` name in the object. See `nis_groups(3N)` for more details.

The `zo_access` member contains the bitmask of access rights assigned to this object. There are four access rights defined, and four are reserved for future use and must be zero. This group of 8 access rights can be granted to four categories of client. These categories are the object's owner, the object's group owner, all authenticated clients (`world`), and all unauthenticated clients (`nobody`). Note that access granted to "nobody" is really access granted to everyone, authenticated and unauthenticated clients.

The `zo_ttl` member contains the number of seconds that the object can "live" in a cache before it is expired. This value is called the time to live for this object. This number is particularly important on group and directory (domain) objects. When an object is cached, the current time is added to the value in `zo_ttl`. Then each time the cached object is used, the time in `zo_ttl` is compared with the current time. If the current time is later than the time in `zo_ttl` the object is said to have expired and the cached copy should not be used.

Setting the TTL is somewhat of an art. You can think of it as the "half life" of the object, or half the amount of time you believe will pass before the object changes. The benefit of setting the `ttl` to a large number is that the object will stay in a cache for long periods of time. The problem with setting it to a large value is that when the object changes it will take a long time for the caches to flush out old copies of that object. The problems and benefits are reversed for setting the time to a small value. Generally setting the value to 43200 (12 hrs) is reasonable for things that change day to day, and 3024000 is good for things that change week to week. Setting the value to 0 will prevent the object from ever being cached since it would expire immediately.

The `zo_data` member is a discriminated union with the following members:

```
zotypes zo_type;
union {
```

```

struct directory_obj di_data;
struct group_obj gr_data;
struct table_obj ta_data;
struct entry_obj en_data;
struct link_obj li_data;
struct {
    uint_t po_data_len;
    char *po_data_val;
} po_data;
} objdata_u;

```

The union is discriminated based on the type value contained in `zo_type`. There six types of objects currently defined in the NIS+ service. These types are the directory, link, group, table, entry, and private types.

```

enum zotypes {
    BOGUS_OBJ = 0,
    NO_OBJ = 1,
    DIRECTORY_OBJ = 2,
    GROUP_OBJ = 3,
    TABLE_OBJ = 4,
    ENTRY_OBJ = 5,
    LINK_OBJ = 6,
    PRIVATE_OBJ = 7
};
typedef enum zotypes zotypes;

```

All object types define a structure that contains data specific to that type of object. The simplest are private objects which are defined to contain a variable length array of octets. Only the owner of the object is expected to understand the contents of a private object. The following section describe the other five object types in more significant detail.

Directory Objects

The first type of object is the *directory* object. This object's variant part is defined as follows:

```

enum nstype {
    UNKNOWN = 0,
    NIS = 1,
    SUNYP = 2,
    DNS = 4,
    X500 = 5,
    DNANS = 6,
    XCHS = 7,
}
typedef enum nstype nstype;
struct oar_mask {
    uint_t oa_rights;
    zotypes oa_otype;
}
typedef struct oar_mask oar_mask;

```

```

struct endpoint {
    char *uaddr;
    char *family;
    char *proto;
}
typedef struct endpoint endpoint;
struct nis_server {
    nis_name name;
    struct {
        uint_t ep_len;
        endpoint *ep_val;
    } ep;
    uint_t key_type;
    netobj pkey;
}
typedef struct nis_server nis_server;
struct directory_obj {
    nis_name do_name;
    nstype do_type;
    struct {
        uint_t do_servers_len;
        nis_server *do_servers_val;
    } do_servers;
    uint32_t do_ttl;
    struct {
        uint_t do_armask_len;
        oar_mask *do_armask_val;
    } do_armask;
}
        typedef struct directory_obj directory_obj;

```

The main structure contains five primary members: `do_name`, `do_type`, `do_servers`, `do_ttl`, and `do_armask`. The information in the `do_servers` structure is sufficient for the client library to create a network connection with the named server for the directory.

The `do_name` member contains the name of the directory or domain represented in a format that is understandable by the type of nameservice serving that domain. In the case of NIS+ domains, this is the same as the name that can be composed using the `zo_name` and `zo_domain` members. For other name services, this name will be a name that they understand. For example, if this were a directory object describing an X.500 namespace that is "under" the NIS+ directory *eng.sun.com.*, this name might contain `"/C=US, /O=Sun Microsystems, /OU=Engineering/"`. The type of nameservice that is being described is determined by the value of the member `do_type`.

The `do_servers` structure contains two members. `do_servers_val` is an array of `nis_server` structures; `do_servers_len` is the number of cells in the array. The `nis_server` structure is designed to contain enough information such that machines on the network providing name services can be contacted without having to use a name service. In the case of NIS+ servers, this

information is the name of the machine in *name*, its public key for authentication in *pkey*, and a variable length array of endpoints, each of which describes the network endpoint for the `rpcbind` daemon on the named machine. The client library uses the addresses to contact the server using a transport that both the client and server can communicate on and then queries the `rpcbind` daemon to get the actual transport address that the server is using.

Note that the first server in the *do_servers* list is always the master server for the directory.

The *key_type* field describes the type of key stored in the *pkey* netobj (see `/usr/include/rpc/xdr.h` for a definition of the network object structure). Currently supported types are `NIS_PK_NONE` for no public key, `NIS_PK_DH` for a Diffie-Hellman type public key, and `NIS_PK_DHEXT` for an extended Diffie-Hellman public key.

The *do_ttl* member contains a copy of the *zo_ttl* member from the common attributes. This is duplicated because the cache manager only caches the variant part of the directory object.

The *do_armask* structure contains two members. *do_armask_val* is an array of *oar_mask* structures; *do_armask_len* is the number of cells in the array. The *oar_mask* structure contains two members: *oa_rights* specifies the access rights allowed for objects of type *oa_otype*. These access rights are used for objects of the given type in the directory when they are present in this array.

The granting of access rights for objects contained within a directory is actually two-tiered. If the directory object itself grants a given access right (using the *zo_access* member in the *nis_object* structure representing the directory), then all objects within the directory are allowed that access. Otherwise, the *do_armask* structure is examined to see if the access is allowed specifically for that type of structure. This allows the administrator of a namespace to set separate policies for different object types, for example, one policy for the creation of tables and another policy for the creation of other directories. See `nis+(1)` for more details.

Link Objects

Link objects provide a means of providing *aliases* or symbolic links within the namespace. Their variant part is defined as follows.

```
struct link_obj {
    zotypes li_rtype;
    struct {
        uint_t li_attrs_len;
        nis_attr *li_attrs_val;
    } li_attrs;
    nis_name li_name;
}
```

The `li_rtype` member contains the object type of the object pointed to by the link. This is only a hint, since the object which the link points to may have changed or been removed. The fully qualified name of the object (table or otherwise) is specified in the member `li_name`.

NIS+ links can point to either other objects within the NIS+ namespace, or to entries within a NIS+ table. If the object pointed to by the link is a table and the member `li_attrs` has a nonzero number of attributes (index name/value pairs) specified, the table is searched when this link is followed. All entries which match the specified search pattern are returned. Note, that unless the flag `FOLLOW_LINKS` is specified, the `nis_lookup(3N)` function will always return non-entry objects.

Group Objects

Group objects contain a membership list of NIS+ principals. The group objects' variant part is defined as follows.

```
struct group_obj {
    uint_t gr_flags;
    struct {
        uint_t gr_members_len;
        nis_name *gr_members_val;
    } gr_members;
}
```

The `gr_flags` member contains flags that are currently unused. The `gr_members` structure contains the list of principals. For a complete description of how group objects are manipulated see `nis_groups(3N)`.

Table Objects

The NIS+ table object is analogous to a YP map. The differences stem from the access controls, and the variable schemas that NIS+ allows. The table objects data structure is defined as follows:

```
#define TA_BINARY 1
#define TA_CRYPT 2
#define TA_XDR 4
#define TA_SEARCHABLE 8
#define TA_CASE 16
#define TA_MODIFIED 32
struct table_col {
    char *tc_name;
    uint_t tc_flags;
    uint_t tc_rights;
}
typedef struct table_col table_col;
struct table_obj {
    char *ta_type;
    uint_t ta_maxcol;
    uchar_t ta_sep;
```

```

struct {
    uint_t ta_cols_len;
    table_col *ta_cols_val;
} ta_cols;
char *ta_path;
}

```

The `ta_type` member contains a string that identifies the type of entries in this table. NIS+ does not enforce any policies as to the contents of this string. However, when entries are added to the table, the NIS+ service will check to see that they have the same “type” as the table as specified by this member.

The structure `ta_cols` contains two members. `ta_cols_val` is an array of `table_col` structures. The length of the array depends on the number of columns in the table; it is defined when the table is created and is stored in `ta_cols_len`. `ta_maxcol` also contains the number of columns in the table and always has the same value as `ta_cols_len`. Once the table is created, this length field cannot be changed.

The `ta_sep` character is used by client applications that wish to print out an entry from the table. Typically this is either space (“ ”) or colon (“:”).

The `ta_path` string defines a concatenation path for tables. This string contains an ordered list of fully qualified table names, separated by colons, that are to be searched if a search on this table fails to match any entries. This path is only used with the flag `FOLLOW_PATH` with a `nis_list()` call. See `nis_tables(3N)` for information on these flags.

In addition to checking the type, the service will check that the number of columns in an entry is the same as those in the table before allowing that entry to be added.

Each column has associated with it a name in `tc_name`, a set of flags in `tc_flags`, and a set of access rights in `tc_rights`. The name should be indicative of the contents of that column.

The `TA_BINARY` flag indicates that data in the column is binary (rather than text). Columns that are searchable cannot contain binary data. The `TA_CRYPT` flag specifies that the information in this column should be encrypted prior to sending it over the network. This flag has no effect in the export version of NIS+. The `TA_XDR` flag is used to tell the client application that the data in this column is encoded using the XDR protocol. The `TA_BINARY` flag must be specified with the XDR flag. Further, by convention, the name of a column that has the `TA_XDR` flag set is the name of the XDR function that will decode the data in that column.

The `TA_SEARCHABLE` flag specifies that values in this column can be searched. Searchable columns must contain textual data and must have a name

Entry Objects

associated with them. The flag `TA_CASE` specifies that searches involving this column ignore the case of the value in the column. At least one of the columns in the table should be searchable. Also, the combination of all searchable column values should uniquely select an entry within the table. The `TA_MODIFIED` flag is set only when the table column is modified. When `TA_MODIFIED` is set, and the object is modified again, the modified access rights for the table column must be copied, not the default access rights.

Entry objects are stored in tables. The structure used to define the entry data is as follows.

```
#define EN_BINARY 1
#define EN_CRYPT 2
#define EN_XDR 4
#define EN_MODIFIED 8
struct entry_col {
    uint_t ec_flags;
    struct {
        uint_t ec_value_len;
        char *ec_value_val;
    } ec_value;
}
typedef struct entry_col entry_col;
struct entry_obj {
    char *en_type;
    struct {
        uint_t en_cols_len;
        entry_col *en_cols_val;
    } en_cols;
}
```

The `en_type` member contains a string that specifies the type of data this entry represents. The NIS+ server will compare this string to the type string specified in the table object and disallow any updates or modifications if they differ.

The `en_cols` structure contains two members: `en_cols_len` and `en_cols_val`. `en_cols_val` is an array of `entry_col` structures. `en_cols_len` contains a count of the number of cells in the `en_cols_val` array and reflects the number of columns in the table – it always contains the same value as the `table_obj.ta_cols.ta_cols_len` member from the table which contains the entry.

The `entry_col` structure contains information about the entry's per-column values. `ec_value` contains information about a particular value. It has two members: `ec_value_val`, which is the value itself, and `ec_value_len`, which is the length (in bytes) of the value. `entry_col` also contains the member `ec_flags`, which contains a set of flags for the entry.

The flags in `ec_flags` are primarily used when adding or modifying entries in a table. All columns that have the flag `EN_CRYPT` set will be encrypted

prior to sending them over the network. Columns with `EN_BINARY` set are presumed to contain binary data. The server will ensure that the column in the table object specifies binary data prior to allowing the entry to be added. When modifying entries in a table, only those columns that have changed need be sent to the server. Those columns should each have the `EN_MODIFIED` flag set to indicate this to the server.

SEE ALSO

`nis+(1)`, `nis_groups(3N)`, `nis_names(3N)`, `nis_server(3N)`,
`nis_subr(3N)`, `nis_tables(3N)`

| | |
|--------------------|---|
| NAME | nis_ping, nis_checkpoint – misc NIS+ log administration functions |
| SYNOPSIS | <pre>cc [flag ...] file ... -lns1 [library ...] #include <rpcsvc/nis.h> void nis_ping(nis_name dirname, uint32_t utime, nis_object * dirobj); nis_result * nis_checkpoint(nis_name dirname);</pre> |
| DESCRIPTION | <p>nis_ping() is called by the master server for a directory when a change has occurred within that directory. The parameter <i>dirname</i> identifies the directory with the change. If the parameter <i>dirobj</i> is <code>NULL</code>, this function looks up the directory object for <i>dirname</i> and uses the list of replicas it contains. The parameter <i>utime</i> contains the timestamp of the last change made to the directory. This timestamp is used by the replicas when retrieving updates made to the directory.</p> <p>The effect of calling nis_ping() is to schedule an update on the replica. A short time after a ping is received, typically about two minutes, the replica compares the last update time for its databases to the timestamp sent by the ping. If the ping timestamp is later, the replica establishes a connection with the master server and request all changes from the log that occurred after the last update that it had recorded in its local log.</p> <p>nis_checkpoint() is used to force the service to checkpoint information that has been entered in the log but has not been checkpointed to disk. When called, this function checkpoints the database for each table in the directory, the database containing the directory and the transaction log. Care should be used in calling this function since directories that have seen a lot of changes may take several minutes to checkpoint. During the checkpointing process, the service will be unavailable for updates for all directories that are served by this machine as master.</p> <p>nis_checkpoint() returns a pointer to a <i>nis_result</i> structure (described in nis_tables(3N)). This structure should be freed with nis_freeresult() (see nis_names(3N)). The only items of interest in the returned result are the status value and the statistics.</p> |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

nislog(1M) , **nis_names(3N)** , **nis_tables(3N)** , **nisfiles(4)** ,
attributes(5)

| | |
|--------------------|---|
| NAME | nis_server, nis_mkdir, nis_rmdir, nis_servstate, nis_stats, nis_getservlist, nis_freeservlist, nis_freetags – miscellaneous NIS+ functions |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] #include <rpcsvc/nis.h> nis_error nis_mkdir(nis_name <i>dirname</i>, nis_server * <i>machine</i>); nis_error nis_rmdir(nis_name <i>dirname</i>, nis_server * <i>machine</i>); nis_error nis_servstate(nis_server * <i>machine</i>, nis_tag * <i>tags</i>, int <i>numtags</i>, nis_tag ** <i>result</i>); nis_error nis_stats(nis_server * <i>machine</i>, nis_tag * <i>tags</i>, int <i>numtags</i>, nis_tag ** <i>result</i>); void nis_freetags(nis_tag * <i>tags</i>, int <i>numtags</i>); nis_server ** nis_getservlist(nis_name <i>dirname</i>); void nis_freeservlist(nis_server ** <i>machines</i>);</pre> |
| DESCRIPTION | <p>These functions provide a variety of services for NIS+ applications.</p> <p>nis_mkdir() is used to create the necessary databases to support NIS+ service for a directory, <i>dirname</i> , on a server, <i>machine</i> . If this operation is successful, it means that the directory object describing <i>dirname</i> has been updated to reflect that server <i>machine</i> is serving the named directory. For a description of the <code>nis_server</code> structure, refer to nis_objects(3N) .</p> <p>nis_rmdir() is used to delete the directory, <i>dirname</i> , from the specified machine. The <i>machine</i> parameter cannot be NULL . For a description of the <code>nis_server</code> structure, refer to nis_objects(3N) .</p> <p>nis_servstate() is used to set and read the various state variables of the NIS+ servers. In particular the internal debugging state of the servers may be set and queried.</p> <p>The nis_stats() function is used to retrieve statistics about how the server is operating. Tracking these statistics can help administrators determine when</p> |

they need to add additional replicas or to break up a domain into two or more subdomains. For more information on reading statistics, see `nisstat(1M)`

`nis_servstate()` and `nis_stats()` use the tag list. This tag list is a variable length array of `nis_tag` structures whose length is passed to the function in the `numtags` parameter. The set of legal tags are defined in the file `<rpcsvc/nis_tags.h>` which is included in `<rpcsvc/nis.h>`. Because these tags can and do vary between implementations of the NIS+ service, it is best to consult this file for the supported list. Passing unrecognized tags to a server will result in their `tag_value` member being set to the string "unknown." Both of these functions return their results in malloced tag structure, `*result`. If there is an error, `*result` is set to `NULL`. The `tag_value` pointers points to allocated string memory which contains the results. Use `nis_freetags()` to free the tag structure.

`nis_getservlist()` returns a null terminated list of `nis_server` structures that represent the list of servers that serve the domain named `dirname`. Servers from this list can be used when calling functions that require the name of a NIS+ server. For a description of the `nis_server` structure, refer to `nis_objects(3N)`. `nis_freeservlist()` frees the list of servers returned by `nis_getservlist()`. Note that this is the only legal way to free that list.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`nisstat(1M)`, `nis_names(3N)`, `nis_objects(3N)`, `nis_subr(3N)`, `attributes(5)`

NAME nis_subr, nis_leaf_of, nis_name_of, nis_domain_of, nis_getnames, nis_freenames, nis_dir_cmp, nis_clone_object, nis_destroy_object, nis_print_object - NIS+ subroutines

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lns1
[
library
... ]
#include <rpcsvc/nis.h>

nis_name nis_leaf_of(const nis_name name);

nis_name nis_name_of(const nis_name name);

nis_name nis_domain_of(const nis_name name);

nis_name * nis_getnames(const nis_name name);

void nis_freenames(nis_name * namelist);

name_pos nis_dir_cmp(const nis_name n1, const nis_name n2);

nis_object * nis_clone_object(const nis_object * src, nis_object * dest);

void nis_destroy_object(nis_object * obj);

void nis_print_object(const nis_object * obj);
```

DESCRIPTION

These subroutines are provided to assist in the development of NIS+ applications. They provide several useful operations on both NIS+ names and objects.

The first group, **nis_leaf_of()**, **nis_domain_of()**, and **nis_name_of()** provide the functions for parsing NIS+ names. **nis_leaf_of()** will return the first label in an NIS+ name. It takes into account the double quote character "" which can be used to protect embedded '.' (dot) characters in object names. Note that the name returned will never have a trailing dot character. If passed the global root directory name ".", it will return the null string.

nis_domain_of() returns the name of the NIS+ domain in which an object resides. This name will always be a fully qualified NIS+ name and ends with a dot. By iteratively calling **nis_leaf_of()** and **nis_domain_of()** it is possible to break a NIS+ name into its individual components.

nis_name_of() is used to extract the unique part of a NIS+ name. This function removes from the tail portion of the name all labels that are in common with the local domain. Thus if a machine were in domain `foo.bar.baz.` and **nis_name_of()** were passed a name `bob.friends.foo.bar.baz`, then **nis_name_of()** would return the unique part, `bob.friends`. If the name passed to this function is not in either the local domain or one of its children, this function will return null.

nis_getnames() will return a list of candidate names for the name passed in as *name*. If this name is not fully qualified, **nis_getnames()** will generate a list of names using the default NIS+ directory search path, or the environment variable `NIS_PATH` if it is set. The returned array of pointers is terminated by a NULL pointer, and the memory associated with this array should be freed by calling `nis_freenames()`.

Though **nis_dir_cmp()** can be used to compare any two NIS+ names, it is used primarily to compare domain names. This comparison is done in a case independent fashion, and the results are an enum of type `name_pos`. When the names passed to this function are identical, the function returns a value of `SAME_NAME`. If the name *n1* is a direct ancestor of name *n2*, then this function returns the result `HIGHER_NAME`. Similarly, if the name *n1* is a direct descendant of name *n2*, then this function returns the result `LOWER_NAME`. When the name *n1* is neither a direct ancestor nor a direct descendant of *n2*, as it would be if the two names were siblings in separate portions of the namespace, then this function returns the result `NOT_SEQUENTIAL`. Finally, if either name cannot be parsed as a legitimate name then this function returns the value `BAD_NAME`.

The second set of functions, consisting of **nis_clone_object()** and **nis_destroy_object()**, are used for manipulating objects. **nis_clone_object()** creates an exact duplicate of the NIS+ object *src*. If the value of *dest* is non-null, it creates the clone of the object into this object structure and allocate the necessary memory for the variable length arrays. If this parameter is null, a pointer to the cloned object is returned. Refer to **nis_objects(3N)** for a description of the `nis_object` structure.

nis_destroy_object() can be used to destroy an object created by **nis_clone_object()**. This will free up all memory associated with the object and free the pointer passed. If the object was cloned into an array (using the *dest* parameter to **nis_clone_object()**) then the object *cannot* be freed with this function. Instead, the function `xdr_free(xdr_nis_object, dest)` must be used.

nis_print_object() prints out the contents of a NIS+ object structure on the standard output. Its primary use is for debugging NIS+ programs.

**ENVIRONMENT
VARIABLES**

NIS_ PATH This variable overrides the default NIS+ directory search path used by `nis_getnames()`. It contains an ordered list of directories separated by ':' (colon) characters. The '\$' (dollar sign) character is treated specially. Directory names that end in '\$' have the default domain appended to them, and a '\$' by itself is replaced by the list of directories between the default domain and the global root that are at least two levels deep. The default NIS+ directory search path is '\$'.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

`nis_names(3N)` , `nis_objects(3N)` , `nis_tables(3N)` , `attributes(5)`

NOTES

`nis_leaf_of()` , `nis_name_of()` and `nis_clone_object()` return their results as thread-specific data in multithreaded applications.

| | |
|--------------------|--|
| NAME | nis_tables, nis_list, nis_add_entry, nis_remove_entry, nis_modify_entry, nis_first_entry, nis_next_entry - NIS+ table functions |
| SYNOPSIS | <pre> cc [flag ...] file ... -lns1 [library ...] #include <rpcsvc/nis.h> nis_result * nis_list(nis_name name, uint_t flags, int (*callback)(nis_name table_name, nis_object * object, void * userdata), void * userdata); nis_result * nis_add_entry(nis_name table_name, nis_object * object, uint_t flags); nis_result * nis_remove_entry(nis_name name, nis_object * object, uint_t flags); nis_result * nis_modify_entry(nis_name name, nis_object * object, uint_t flags); nis_result * nis_first_entry(nis_name table_name); nis_result * nis_next_entry(nis_name table_name, netobj * cookie); void nis_freeresult(nis_result * result); </pre> |
| DESCRIPTION | <p>These functions are used to search and modify NIS+ tables. nis_list() is used to search a table in the NIS+ namespace. nis_first_entry() and nis_next_entry() are used to enumerate a table one entry at a time. nis_add_entry() , nis_remove_entry() , and nis_modify_entry() are used to change the information stored in a table. nis_freeresult() is used to free the memory associated with the <code>nis_result</code> structure.</p> <p>Entries within a table are named by NIS+ indexed names. An indexed name is a compound name that is composed of a search criteria and a simple NIS+ name that identifies a table object. A search criteria is a series of column names and their associated values enclosed in bracket '[''] characters. Indexed names have the following form:</p> <pre> [colname = value </pre> |

```

, . . . ],
tablename

```

The list function, **nis_list()**, takes an indexed name as the value for the *name* parameter. Here, the *tablename* should be a fully qualified NIS+ name unless the **EXPAND_NAME** flag (described below) is set. The second parameter, *flags*, defines how the function will respond to various conditions. The value for this parameter is created by logically OR ing together one or more flags from the following list.

FOLLOW_LINKS If the table specified in *name* resolves to be a **LINK** type object (see **nis_objects(3N)**), this flag specifies that the client library follow that link and do the search at that object. If this flag is not set and the name resolves to a link, the error **NIS_NOTSEARCHABLE** will be returned.

FOLLOW_PATH This flag specifies that if the entry is not found within this table, the list operation should follow the path specified in the table object. When used in conjunction with the **ALL_RESULTS** flag below, it specifies that the path should be followed regardless of the result of the search. When used in conjunction with the **FOLLOW_LINKS** flag above, named tables in the path that resolve to links will be followed until the table they point to is located. If a table in the path is not reachable because no server that serves it is available, the result of the operation will be either a “soft” success or a “soft” failure to indicate that not all tables in the path could be searched. If a name in the path names is either an invalid or non-existent object then it is silently ignored.

HARD_LOOKUP This flag specifies that the operation should continue trying to contact a server of the named table until a definitive result is returned (such as **NIS_NOTFOUND**).

ALL_RESULTS This flag can only be used in conjunction with **FOLLOW_PATH** and a callback function. When specified, it forces all of the tables in the path to be searched. If *name* does not specify a search criteria (imply that all entries are to be returned), then this flag will cause all of the entries in all of the tables in the path to be returned.

NO_CACHE This flag specifies that the client library should bypass any client object caches and get its information directly from

either the master server or a replica server for the named table.

MASTER_ONLY This flag is even stronger than `NO_CACHE` in that it specifies that the client library should *only* get its information from the master server for a particular table. This guarantees that the information will be up to date. However, there may be severe performance penalties associated with contacting the master server directly on large networks. When used in conjunction with the `HARD_LOOKUP` flag, this will block the list operation until the master server is up and available.

EXPAND_NAME When specified, the client library will attempt to expand a partially qualified name by calling `nis_getnames()` (see `nis_local_names(3N)`) which uses the environment variable `NIS_PATH`.

RETURN_RESULT This flag is used to specify that a copy of the returning object be returned in the `nis_result` structure if the operation was successful.

The third parameter to `nis_list()`, `callback`, is an optional pointer to a function that will process the `ENTRY` type objects that are returned from the search. If this pointer is `NULL`, then all entries that match the search criteria are returned in the `nis_result` structure, otherwise this function will be called once for each entry returned. When called, this function should return 0 when additional objects are desired and 1 when it no longer wishes to see any more objects. The fourth parameter, `userdata`, is simply passed to callback function along with the returned entry object. The client can use this pointer to pass state information or other relevant data that the callback function might need to process the entries.

The `nis_list()` function is not MT-Safe with callbacks. See `NOTES`.

`nis_add_entry()` will add the NIS+ object to the NIS+ `table_name`. The `flags` parameter is used to specify the failure semantics for the add operation. The default (`flags` equal 0) is to fail if the entry being added already exists in the table. The `ADD_OVERWRITE` flag may be used to specify that existing object is to be overwritten if it exists, (a modify operation) or added if it does not exist. With the `ADD_OVERWRITE` flag, this function will fail with the error `NIS_PERMISSION` if the existing object does not allow modify privileges to the client.

If the flag `RETURN_RESULT` has been specified, the server will return a copy of the resulting object if the operation was successful.

`nis_remove_entry()` removes the identified entry from the table or a set of entries identified by `table_name`. If the parameter `object` is non-null, it is

presumed to point to a cached copy of the entry. When the removal is attempted, and the object that would be removed is not the same as the cached object pointed to by *object* then the operation will fail with an `NIS_NOTSAMEOBJ` error. If an object is passed with this function, the search criteria in *name* is optional as it can be constructed from the values within the entry. However, if no object is present, the search criteria must be included in the *name* parameter. If the flags variable is null, and the search criteria does not uniquely identify an entry, the `NIS_NOTUNIQUE` error is returned and the operation is aborted. If the flag parameter `REM_MULTIPLE` is passed, and if remove permission is allowed for each of these objects, then all objects that match the search criteria will be removed. Note that a null search criteria and the `REM_MULTIPLE` flag will remove all entries in a table.

nis_modify_entry() modifies an object identified by *name* . The parameter *object* should point to an entry with the `EN_MODIFIED` flag set in each column that contains new information.

The owner, group, and access rights of an entry are modified by placing the modified information into the respective fields of the parameter, *object* :
`zo_owner` , `zo_group` , and `zo_access` .

These columns will replace their counterparts in the entry that is stored in the table. The entry passed must have the same number of columns, same type, and valid data in the modified columns for this operation to succeed.

If the flags parameter contains the flag `MOD_SAMEOBJ` then the object pointed to by *object* is assumed to be a cached copy of the original object. If the OID of the object passed is different than the OID of the object the server fetches, then the operation fails with the `NIS_NOTSAMEOBJ` error. This can be used to implement a simple read-modify-write protocol which will fail if the object is modified before the client can write the object back.

If the flag `RETURN_RESULT` has been specified, the server will return a copy of the resulting object if the operation was successful.

nis_first_entry() fetches entries from a table one at a time. This mode of operation is extremely inefficient and callbacks should be used instead wherever possible. The table containing the entries of interest is identified by *name* . If a search criteria is present in *name* it is ignored. The value of *cookie* within the `nis_result` structure must be copied by the caller into local storage and passed as an argument to **nis_next_entry()** .

nis_next_entry() retrieves the “next” entry from a table specified by *table_name* . The order in which entries are returned is not guaranteed. Further, should an update occur in the table between client calls to **nis_next_entry()** there is no guarantee that an entry that is added or modified will be seen by the client. Should an entry be removed from the table that would have been the “next” entry returned, the error `NIS_CHAINBROKEN` is returned instead.

RETURN VALUES

These functions return a pointer to a structure of type `nis_result` :

```
struct nis_result {
  \011\011nis_error\011status;
  \011\011struct {
    \011\011\011uint_t\011objects_len;
    \011\011\011nis_object\011*objects_val;
    \011\011} objects;
    \011\011netobj\011cookie;
    \011\011uint32_t\011zticks;
    \011\011uint32_t\011dticks;
    \011\011uint32_t\011aticks;
    \011\011uint32_t\011cticks;
  };
```

The *status* member contains the error status of the the operation. A text message that describes the error can be obtained by calling the function `nis_sperno()` (see `nis_error(3N)`).

The *objects* structure contains two members. *objects_val* is an array of *nis_object* structures; *objects_len* is the number of cells in the array. These objects will be freed by a call to `nis_freeresult()` (see `nis_names(3N)`). If you need to keep a copy of one or more objects, they can be copied with the function `nis_clone_object()` and freed with the function `nis_destroy_object()` (see `nis_server(3N)`).

The various ticks contain details of where the time (in microseconds) was taken during a request. They can be used to tune one's data organization for faster access and to compare different database implementations (see `nis_db(3N)`).

zticks The time spent in the NIS+ service itself, this count starts when the server receives the request and stops when it sends the reply.

dticks The time spent in the database backend, this time is measured from the time a database call starts, until a result is returned. If the request results in multiple calls to the database, this is the sum of all the time spent in those calls.

aticks The time spent in any "accelerators" or caches. This includes the time required to locate the server needed to resolve the request.

cticks The total time spent in the request, this clock starts when you enter the client library and stops when a result is returned. By subtracting the sum of the other ticks values from this value you can obtain the local overhead of generating a NIS+ request.

Subtracting the value in *dticks* from the value in *zticks* will yield the time spent in the service code itself. Subtracting the sum of the values in *zticks* and *aticks*

from the value in *cticks* will yield the time spent in the client library itself.
Note: all of the tick times are measured in microseconds.

ERRORS

The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.

| | |
|------------------|--|
| NIS_BADATTRIBUTE | The name of an attribute did not match up with a named column in the table, or the attribute did not have an associated value. |
| NIS_BADNAME | The name passed to the function is not a legal NIS+ name. |
| NIS_BADREQUEST | A problem was detected in the request structure passed to the client library. |
| NIS_CACHEEXPIRED | The entry returned came from an object cache that has <i>expired</i> . This means that the time to live value has gone to zero and the entry may have changed. If the flag NO_CACHE was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object. |
| NIS_CBERROR | An RPC error occurred on the server while it was calling back to the client. The transaction was aborted at that time and any unsent data was discarded. |
| NIS_CBRESULTS | Even though the request was successful, all of the entries have been sent to your callback function and are thus not included in this result. |
| NIS_FOREIGNNS | The name could not be completely resolved. When the name passed to the function would resolve in a namespace that is outside the NIS+ name tree, this error is returned with a NIS+ object of type DIRECTORY. The returned object contains the type of namespace and contact information for a server within that namespace. |
| NIS_INVALIDOBJ | The object pointed to by <i>object</i> is not a valid NIS+ entry object for the given table. This could occur if it had a mismatched number of columns, or a different data type (for example, binary or text) than the associated column in the table. |

| | |
|---------------------|---|
| NIS_LINKNAMEERROR | The name passed resolved to a LINK type object and the contents of the object pointed to an invalid name. |
| NIS_MODFAIL | The attempted modification failed for some reason. |
| NIS_NAMEEXISTS | An attempt was made to add a name that already exists. To add the name, first remove the existing name and then add the new name or modify the existing named object. |
| NIS_NAMEUNREACHABLE | This soft error indicates that a server for the desired directory of the named table object could not be reached. This can occur when there is a network partition or the server has crashed. Attempting the operation again may succeed. See the HARD_LOOKUP flag. |
| NIS_NOCALLBACK | The server was unable to contact the callback service on your machine. This results in no data being returned. |
| NIS_NOMEMORY | Generally a fatal result. It means that the service ran out of heap space. |
| NIS_NOSUCHNAME | This hard error indicates that the named directory of the table object does not exist. This occurs when the server that should be the parent of the server that serves the table, does not know about the directory in which the table resides. |
| NIS_NOSUCHTABLE | The named table does not exist. |
| NIS_NOT_ME | A request was made to a server that does not serve the given name. Normally this will not occur, however if you are not using the built in location mechanism for servers, you may see this if your mechanism is broken. |
| NIS_NOTFOUND | No entries in the table matched the search criteria. If the search criteria was null (return all entries) then this result means that the table is empty and may safely be removed by calling the nis_remove() . |

| | |
|-------------------|---|
| | <p>If the FOLLOW_PATH flag was set, this error indicates that none of the tables in the path contain entries that match the search criteria.</p> |
| NIS_NOTMASTER | <p>A change request was made to a server that serves the name, but it is not the master server. This can occur when a directory object changes and it specifies a new master server. Clients that have cached copies of the directory object in the <code>/var/nis/NIS_SHARED_DIRCACHE</code> file will need to have their cache managers restarted (use <code>nis_cachemgr -i</code>) to flush this cache.</p> |
| NIS_NOTSAMEOBJ | <p>An attempt to remove an object from the namespace was aborted because the object that would have been removed was not the same object that was passed in the request.</p> |
| NIS_NOTSEARCHABLE | <p>The table name resolved to a NIS+ object that was not searchable.</p> |
| NIS_PARTIAL | <p>This result is similar to NIS_NOTFOUND except that it means the request succeeded but resolved to zero entries. When this occurs, the server returns a copy of the table object instead of an entry so that the client may then process the path or implement some other local policy.</p> |
| NIS_RPCERROR | <p>This fatal error indicates the RPC subsystem failed in some way. Generally there will be a <code>syslog(3)</code> message indicating why the RPC request failed.</p> |
| NIS_S_NOTFOUND | <p>The named entry does not exist in the table, however not all tables in the path could be searched, so the entry may exist in one of those tables.</p> |
| NIS_S_SUCCESS | <p>Even though the request was successful, a table in the search path was not able to be searched, so the result may not be the same as the one you would have received if that table had been accessible.</p> |
| NIS_SUCCESS | <p>The request was successful.</p> |

| | |
|------------------|---|
| NIS_SYSTEMERROR | Some form of generic system error occurred while attempting the request. Check the <code>syslog(3)</code> record for error messages from the server. |
| NIS_TOOMANYATTRS | The search criteria passed to the server had more attributes than the table had searchable columns. |
| NIS_TRYAGAIN | The server connected to was too busy to handle your request. <code>add_entry()</code> , <code>remove_entry()</code> , and <code>modify_entry()</code> return this error when the master server is currently updating its internal state. It can be returned to <code>nis_list()</code> when the function specifies a callback and the server does not have the resources to handle callbacks. |
| NIS_TYPEMISMATCH | An attempt was made to add or modify an entry in a table, and the entry passed was of a different type than the table. |

ENVIRONMENT VARIABLES

| | |
|----------|---|
| NIS_PATH | When set, this variable is the search path used by <code>nis_list()</code> if the flag <code>EXPAND_NAME</code> is set. |
|----------|---|

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |

SEE ALSO

`niscat(1)`, `niserror(1)`, `nismatch(1)`, `nis_cachemgr(1M)`, `nis_clone_object(3N)`, `nis_db(3N)`, `nis_destroy_object(3N)`, `nis_error(3N)`, `nis_getnames(3N)`, `nis_local_names(3N)`, `nis_names(3N)`, `nis_objects(3N)`, `nis_server(3N)`, `rpc_svc_calls(3N)`, `syslog(3)`, `attributes(5)`

WARNINGS

Use the flag `HARD_LOOKUP` carefully since it can cause the application to block indefinitely during a network partition.

NOTES

The path used when the flag `FOLLOW_PATH` is specified, is the one present in the *first* table searched. The path values in tables that are subsequently searched are ignored.

It is legal to call functions that would access the nameservice from within a list callback. However, calling a function that would itself use a callback, or calling

nis_list() with a callback from within a list callback function is not currently supported.

There are currently no known methods for **nis_first_entry()** and **nis_next_entry()** to get their answers from only the master server.

The **nis_list()** function is not MT-Safe with callbacks. **nis_list()** callbacks are serialized. A call to **nis_list()** with a callback from within **nis_list()** will deadlock. **nis_list()** with a callback cannot be called from an rpc server. See **rpc_svc_calls(3N)** . Otherwise, this function is MT-Safe.

| | |
|----------------------|---|
| NAME | nl, nonl – enable/disable newline control |
| SYNOPSIS | <pre>#include <curses.h> int nl(void); int nonl(void);</pre> |
| DESCRIPTION | <p>The nl() function enables the handling of newlines. The nl() function converts newline into carriage return and line feed on output and converts carriage return into newline on input. nonl() disables the handling of newlines.</p> <p>The handling of newlines is initially enabled. Disabling the handling of newlines results in faster cursor motion since X/Open Curses can use the line-feed capability more efficiently.</p> |
| RETURN VALUES | On success, these functions return <code>OK</code> . Otherwise, they return <code>ERR</code> . |
| ERRORS | None. |

| | |
|----------------------|---|
| NAME | nlist – get entries from symbol table |
| SYNOPSIS | <pre>/usr/ucb/cc [flag ...] file ... #include <nlist.h> int nlist(filename, nl); char *filename; struct nlist *nl;</pre> |
| DESCRIPTION | <p>nlist() examines the symbol table from the executable image whose name is pointed to by <i>filename</i>, and selectively extracts a list of values and puts them in the array of <code>nlist</code> structures pointed to by <i>nl</i>. The name list pointed to by <i>nl</i> consists of an array of structures containing names, types and values. The <code>n_name</code> field of each such structure is taken to be a pointer to a character string representing a symbol name. The list is terminated by an entry with a <code>NULL</code> pointer (or a pointer to a <code>NULL</code> string) in the <code>n_name</code> field. For each entry in <i>nl</i>, if the named symbol is present in the executable image's symbol table, its value and type are placed in the <code>n_value</code> and <code>n_type</code> fields. If a symbol cannot be located, the corresponding <code>n_type</code> field of <i>nl</i> is set to zero.</p> |
| RETURN VALUES | Upon normal completion, nlist() returns the number of symbols that were not located in the symbol table. If an error occurs, nlist() returns <code>-1</code> and sets all of the <code>n_type</code> fields in members of the array pointed to by <i>nl</i> to zero. |
| SEE ALSO | <code>nlist(3E)</code> , <code>a.out(4)</code> |
| NOTES | <p>Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.</p> <p>Only the <code>n_value</code> field is compatibly set. Other fields in the <code>nlist</code> structure are filled with the ELF (Executable and Linking Format) values (see <code>nlist(3E)</code> and <code>a.out(4)</code>).</p> |

| NAME | nlist - get entries from name list | | | | |
|----------------------|--|----------------|-----------------|----------|------|
| SYNOPSIS | <pre>cc [<i>flag...</i>] <i>file</i> ... -lelf [<i>library</i> ...] #include <nlist.h> int nlist(const char *<i>filename</i>, struct nlist *<i>nl</i>);</pre> | | | | |
| DESCRIPTION | <p>nlist() examines the name list in the executable file whose name is pointed to by <i>filename</i>, and selectively extracts a list of values and puts them in the array of nlist() structures pointed to by <i>nl</i>. The name list <i>nl</i> consists of an array of structures containing names of variables, types, and values. The list is terminated with a null name, that is, a null string is in the name position of the structure. Each variable name is looked up in the name list of the file. If the name is found, the type, value, storage class, and section number of the name are inserted in the other fields. The type field may be set to 0 if the file was not compiled with the <code>-g</code> option to <code>cc(1B)</code>.</p> <p>nlist() will always return the information for an external symbol of a given name if the name exists in the file. If an external symbol does not exist, and there is more than one symbol with the specified name in the file (such as static symbols defined in separate files), the values returned will be for the last occurrence of that name in the file. If the name is not found, all fields in the structure except <code>n_name</code> are set to 0.</p> <p>This function is useful for examining the system name list kept in the file <code>/dev/ksyms</code>. In this way programs can obtain system addresses that are up to date.</p> | | | | |
| RETURN VALUES | <p>All value entries are set to 0 if the file cannot be read or if it does not contain a valid name list.</p> <p>nlist() returns 0 on success, -1 on error.</p> | | | | |
| ATTRIBUTES | <p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe | | | | |
| SEE ALSO | <p><code>cc(1B)</code>, <code>elf(3E)</code>, <code>kvm_nlist(3K)</code>, <code>kvm_open(3K)</code>, <code>a.out(4)</code>, <code>attributes(5)</code>, <code>ksyms(7D)</code>, <code>mem(7D)</code></p> | | | | |

| NAME | nl_langinfo - language information | | | | | | |
|----------------------|--|----------------|-----------------|----------|-------------------------|-----|---------|
| SYNOPSIS | <pre>#include <langinfo.h> char *nl_langinfo(nl_item item);</pre> | | | | | | |
| DESCRIPTION | <p>The nl_langinfo() function returns a pointer to a null-terminated string containing information relevant to a particular language or cultural area defined in the programs locale. The manifest constant names and values of <i>item</i> are defined by <langinfo.h>. For example:</p> <pre>nl_langinfo (ABDAY_1);</pre> <p>would return a pointer to the string “Dim” if the identified language was French and a French locale was correctly installed; or “Sun” if the identified language was English.</p> | | | | | | |
| RETURN VALUES | <p>If setlocale(3C) has not been called successfully, or if data for a supported language is either not available, or if <i>item</i> is not defined therein, then nl_langinfo() returns a pointer to the corresponding string in the C locale. In all locales, nl_langinfo() returns a pointer to an empty string if <i>item</i> contains an invalid setting.</p> | | | | | | |
| USAGE | <p>The nl_langinfo() function can be used safely in multithreaded applications, as long as setlocale(3C) is not being called to change the locale.</p> | | | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" data-bbox="485 1215 1385 1346"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe with exceptions</td> </tr> <tr> <td>CSI</td> <td>Enabled</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe with exceptions | CSI | Enabled |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| MT-Level | MT-Safe with exceptions | | | | | | |
| CSI | Enabled | | | | | | |
| SEE ALSO | setlocale(3C) , attributes(5) , langinfo(5) , nl_types(5) | | | | | | |
| WARNINGS | <p>The array pointed to by the return value should not be modified by the program. Subsequent calls to nl_langinfo() may overwrite the array.</p> | | | | | | |

| NAME | nlsgetcall – get client’s data passed via the listener | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>#include <sys/tiuser.h> struct t_call *nlsgetcall(int <i>fildes</i>);</pre> | | | | |
| DESCRIPTION | <p>nlsgetcall() allows server processes started by the listener process to access the client’s <code>t_call</code> structure, that is, the <code>sndcall</code> argument of <code>t_connect(3N)</code>.</p> <p>The <code>t_call</code> structure returned by nlsgetcall() can be released using <code>t_free(3N)</code>.</p> <p>nlsgetcall() returns the address of an allocated <code>t_call</code> structure or NULL if a <code>t_call</code> structure cannot be allocated. If the <code>t_alloc()</code> succeeds, undefined environment variables are indicated by a negative <i>len</i> field in the appropriate <code>netbuf</code> structure. A <i>len</i> field of zero in the <code>netbuf</code> structure is valid and means that the original buffer in the listener’s <code>t_call</code> structure was NULL.</p> | | | | |
| RETURN VALUES | A NULL pointer is returned if a <code>t_call</code> structure cannot be allocated by <code>t_alloc()</code> . <code>t_errno</code> can be inspected for further error information. Undefined environment variables are indicated by a negative length field (<i>len</i>) in the appropriate <code>netbuf</code> structure. | | | | |
| FILES | <pre>/usr/lib/libnsl_s.a /usr/lib/libslan.a /usr/lib/libnls.a</pre> | | | | |
| ATTRIBUTES | See <code>attributes(5)</code> for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | <code>nlsadmin(1M)</code> , <code>getenv(3C)</code> , <code>t_alloc(3N)</code> , <code>t_connect(3N)</code> , <code>t_error(3N)</code> , <code>t_free(3N)</code> , <code>t_sync(3N)</code> , <code>attributes(5)</code> | | | | |
| WARNINGS | <p>The <i>len</i> field in the <code>netbuf</code> structure is defined as being unsigned. In order to check for error returns, it should first be cast to an int.</p> <p>The listener process limits the amount of user data (<i>udata</i>) and options data (<i>opt</i>) to 128 bytes each. Address data <i>addr</i> is limited to 64 bytes. If the original data was longer, no indication of overflow is given.</p> | | | | |

NOTES

Server processes must call `t_sync(3N)` before calling this routine.

This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

| NAME | nlsprovider – get name of transport provider | | | | |
|----------------------|---|----------------|-----------------|----------|--------|
| SYNOPSIS | char *nlsprovider(void); | | | | |
| DESCRIPTION | <p>nlsprovider() returns a pointer to a null-terminated character string which contains the name of the transport provider as placed in the environment by the listener process. If the variable is not defined in the environment, a NULL pointer is returned.</p> <p>The environment variable is only available to server processes started by the listener process.</p> | | | | |
| RETURN VALUES | If the variable is not defined in the environment, a NULL pointer is returned. | | | | |
| FILES | <p>/usr/lib/libslan.a (7300)</p> <p>/usr/lib/libnls.a (3B2 Computer)</p> <p>/usr/lib/libnsl_s.a</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | nlsadmin(1M) , attributes(5) | | | | |
| NOTES | This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread. | | | | |

| | |
|----------------------|---|
| NAME | nlsrequest – format and send listener service request message |
| SYNOPSIS | <pre>#include <listen.h> int nlsrequest(int <i>fildev</i>, char *<i>service_code</i>); extern int _nlslogt_errno; extern char *_nlsrmsg;</pre> |
| DESCRIPTION | <p>Given a virtual circuit to a listener process (<i>fildev</i>) and a service code of a server process, nlsrequest() formats and sends a <i>service request message</i> to the remote listener process requesting that it start the given service. nlsrequest() waits for the remote listener process to return a <i>service request response message</i>, which is made available to the caller in the static, null-terminated data buffer pointed to by <i>_nlsrmsg</i>. The <i>service request response message</i> includes a success or failure code and a text message. The entire message is printable.</p> |
| RETURN VALUES | <p>The success or failure code is the integer return code from nlsrequest(). Zero indicates success, other negative values indicate nlsrequest() failures as follows:</p> <ul style="list-style-type: none"> -1 Error encountered by nlsrequest(), see <i>t_errno</i>. <p>Positive values are error return codes from the <i>listener</i> process. Mnemonics for these codes are defined in <i><listen.h></i>.</p> <ul style="list-style-type: none"> 2 Request message not interpretable. 3 Request service code unknown. 4 Service code known, but currently disabled. <p>If non-null, <i>_nlsrmsg</i> contains a pointer to a static, null-terminated character buffer containing the <i>service request response message</i>. Note that both <i>_nlsrmsg</i> and the data buffer are overwritten by each call to nlsrequest().</p> <p>If <i>_nlslog</i> is non-zero, nlsrequest() prints error messages on <i>stderr</i>. Initially, <i>_nlslog</i> is zero.</p> |
| FILES | <pre>/usr/lib/libnls.a /usr/lib/libslan.a /usr/lib/libnsl_s.a</pre> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `nlsadmin(1M)`, `t_error(3N)`, `t_snd(3N)`, `t_rcv(3N)`, `attributes(5)`

WARNINGS `nlsrequest()` cannot always be certain that the remote server process has been successfully started. In this case, `nlsrequest()` returns with no indication of an error and the caller will receive notification of a disconnect event by way of a `T_LOOK` error before or during the first `t_snd()` or `t_rcv()` call.

NOTES These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

| | |
|----------------------|---|
| NAME | nodelay – set blocking or non-blocking read |
| SYNOPSIS | <pre>#include <curses.h> int nodelay(WINDOW *win, bool bf);</pre> |
| PARAMETERS | <p>win Is a pointer to the window in which to enable non-blocking.</p> <p>bf Is a Boolean expression.</p> |
| DESCRIPTION | If enabled, (<i>bf</i> is TRUE), the nodelay() function causes getch(3XC) to return ERR if no input is ready. When disabled, getch() blocks until a key is pressed. |
| RETURN VALUES | On success, the nodelay() function returns OK. Otherwise, it returns ERR. |
| ERRORS | None. |
| SEE ALSO | getch(3XC) , halfdelay(3XC) , notimeout(3XC) |

| | |
|----------------------|--|
| NAME | noqiflush, qiflush – control flush of input and output on interrupt |
| SYNOPSIS | <pre>#include <curses.h> void noqiflush(void); void qiflush(void);</pre> |
| DESCRIPTION | The qiflush() function enables the flushing of input and output queues when an interrupt, quit, or suspend character is sent to the terminal. The noqiflush() function disables this flushing. |
| RETURN VALUES | These functions do not return a value. |
| ERRORS | None |
| SEE ALSO | flushinp(3XC) , intrflush(3XC) |

| | |
|--------------------|---|
| NAME | NOTE, _NOTE – annotate source code with info for tools |
| SYNOPSIS | <pre>#include <note.h> NOTE(NoteInfo); or #include<sys/note.h> _NOTE(NoteInfo);</pre> |
| DESCRIPTION | <p>These macros are used to embed information for tools in program source. A use of one of these macros is called an “annotation”. A tool may define a set of such annotations which can then be used to provide the tool with information that would otherwise be unavailable from the source code.</p> <p>Annotations should, in general, provide documentation useful to the human reader. If information is of no use to a human trying to understand the code but is necessary for proper operation of a tool, use another mechanism for conveying that information to the tool (one which does not involve adding to the source code), so as not to detract from the readability of the source. The following is an example of an annotation which provides information of use to a tool and to the human reader (in this case, which data are protected by a particular lock, an annotation defined by the static lock analysis tool <code>lock_lint</code>).</p> <pre>NOTE(MUTEX_PROTECTS_DATA(foo_lock, foo_list Foo))</pre> <p>Such annotations do not represent executable code; they are neither statements nor declarations. They should not be followed by a semicolon. If a compiler or tool that analyzes C source does not understand this annotation scheme, then the tool will ignore the annotations. (For such tools, <code>NOTE(x)</code> expands to nothing.)</p> <p>Annotations may only be placed at particular places in the source. These places are where the following C constructs would be allowed:</p> <ul style="list-style-type: none"> ■ a top-level declaration (that is, a declaration not within a function or other construct) ■ a declaration or statement within a block (including the block which defines a function) ■ a member of a <code>struct</code> or <code>union</code> . <p>Annotations are not allowed in any other place. For example, the following are illegal:</p> <pre>x = y + NOTE(...) z ; typedef NOTE(...) unsigned int uint ;</pre> |

While `NOTE` and `_NOTE` may be used in the places described above, a particular type of annotation may only be allowed in a subset of those places. For example, a particular annotation may not be allowed inside a `struct` or `union` definition.

NOTE vs _NOTE

Ordinarily, `NOTE` should be used rather than `_NOTE`, since use of `_NOTE` technically makes a program non-portable. However, it may be inconvenient to use `NOTE` for this purpose in existing code if `NOTE` is already heavily used for another purpose. In this case one should use a different macro and write a header file similar to `/usr/include/note.h` which maps that macro to `_NOTE` in the same manner. For example, the following makes `FOO` such a macro:

```
#ifndef _FOO_H
#define _FOO_H
#define FOO _NOTE
#include <sys/note.h>
#endif
```

Public header files which span projects should use `_NOTE` rather than `NOTE`, since `NOTE` may already be used by a program which needs to include such a header file.

NoteInfo Argument

The actual *NoteInfo* used in an annotation should be specified by a tool that deals with program source (see the documentation for the tool to determine which annotations, if any, it understands).

NoteInfo must have one of the following forms:

```
NoteName
NoteName
(
  Args
)
```

where *NoteName* is simply an identifier which indicates the type of annotation, and *Args* is something defined by the tool that specifies the particular

NoteName. The general restrictions on *Args* are that it be compatible with an ANSI C tokenizer and that unquoted parentheses be balanced (so that the end of the annotation can be determined without intimate knowledge of any particular annotation).

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

note(4) , **attributes(5)**

| | |
|----------------------|--|
| NAME | notimeout, timeout, wtimeout – set timed blocking or non-blocking read |
| SYNOPSIS | <pre>#include <curses.h> int notimeout(WINDOW * win, bool bf); void timeout(int delay); void wtimeout(WINDOW win, int delay);</pre> |
| PARAMETERS | <p>win Is a pointer to the window in which to set the timed blocking.</p> <p>bf Is a Boolean expression.</p> <p>delay Is the number of milliseconds to block or wait for input.</p> |
| DESCRIPTION | <p>If <i>bool</i> is TRUE , the notimeout() function disables a timer used by getch(3XC) when handling multibyte function key sequences.</p> <p>When <i>bool</i> is FALSE and keypad handling is enabled, a timer is set by getch() to handle bytes received that could be the beginning of a function key (for example, ESC). If the remainder of the sequence is not received before the time expires, the first byte is returned; otherwise, the value of the function key is returned. Subsequent calls to the getch() function will return the other bytes received for the incomplete key sequence.</p> <p>The timeout() and wtimeout() functions set the length of time getch() waits for input for windows <code>stdscr</code> and <i>win</i> , respectively. These functions are similar to nodelay(3XC) except the time to block or wait for input can be specified.</p> <p>A negative <i>delay</i> causes the program to wait indefinitely for input; a <i>delay</i> of 0 returns ERR if no input is ready; and a positive <i>delay</i> blocks until input arrives or the time specified expires, (in which case, ERR is returned).</p> |
| RETURN VALUES | <p>On success, the notimeout() function returns OK . Otherwise, it returns ERR .</p> <p>The timeout() and wtimeout() functions do not return a value.</p> |
| ERRORS | None. |
| SEE ALSO | getch(3XC) , halfdelay(3XC) , nodelay(3XC) |

NAME `offsetof` – offset of structure member

SYNOPSIS `#include <stddef.h>`
 `size_t offsetof(type, member-designator);`

DESCRIPTION The **offsetof()** macro defined in `<stddef.h>` expands to an integral constant expression that has type `size_t`. The value of this expression is the offset in bytes to the structure member (designated by *member-designator*) from the beginning of its structure (designated by *type*).

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO `attributes(5)`

| | |
|----------------------|--|
| NAME | opendir – open directory |
| SYNOPSIS | <pre>#include <sys/types.h> #include <dirent.h> DIR *opendir(const char *dirname);</pre> |
| DESCRIPTION | <p>The opendir() function opens a directory stream corresponding to the directory named by the <code>dirname</code> argument. The directory stream is positioned at the first entry. If the type <code>DIR</code>, is implemented using a file descriptor, applications will only be able to open up to a total of <code>OPEN_MAX</code> files and directories. A successful call to any of the <code>exec</code> functions will close any directory streams that are open in the calling process. See exec(2).</p> |
| RETURN VALUES | <p>Upon successful completion, opendir() returns a pointer to an object of type <code>DIR</code>. Otherwise, a null pointer is returned and <code>errno</code> is set to indicate the error.</p> |
| ERRORS | <p>The opendir() function will fail if:</p> <p>EACCES Search permission is denied for the component of the path prefix of <code>dirname</code> or read permission is denied for <code>dirname</code>.</p> <p>ELOOP Too many symbolic links were encountered in resolving <code>path</code>.</p> <p>ENAMETOOLONG The length of the <code>dirname</code> argument exceeds <code>PATH_MAX</code>, or a pathname component is longer than <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.</p> <p>ENOENT A component of <code>dirname</code> does not name an existing directory or <code>dirname</code> is an empty string.</p> <p>ENOTDIR A component of <code>dirname</code> is not a directory.</p> <p>The opendir() function may fail if:</p> <p>EMFILE There are <code>OPEN_MAX</code> file descriptors currently open in the calling process.</p> <p>ENAMETOOLONG Pathname resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code>.</p> <p>ENFILE Too many files are currently open in the system.</p> |

USAGE The **opendir()** function should be used in conjunction with **readdir(3C)**, **closedir(3C)** and **rewinddir(3C)** to examine the contents of the directory (see the **EXAMPLES** section in **readdir(3C)**). This method is recommended for portability.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO **lstat(2)**, **symlink(2)**, **closedir(3C)**, **readdir(3C)**, **rewinddir(3C)**, **attributes(5)**

| | |
|--------------------|--|
| NAME | overlay, overwrite – overlap or overwrite windows |
| SYNOPSIS | <pre>#include <curses.h> int overlay(WINDOW *const srcwin, WINDOW * dstwin); int overwrite(WINDOW *const srcwin, WINDOW * dstwin);</pre> |
| PARAMETERS | <p>srcwin Is a pointer to the source window to be copied.</p> <p>dstwin Is a pointer to the destination window to be overlaid or overwritten.</p> |
| DESCRIPTION | <p>The overwrite() and overlay() functions copy the overlapping portion of <i>srcwin</i> to <i>dstwin</i> . <i>srcwin</i> and <i>dstwin</i> do not have to be the same size.</p> <p>The overwrite() function copies the characters from the overlapping portion to <i>dstwin</i> ; thus, destroying the previous contents of the window. The overlay() function only copies non-blank characters, leaving blank characters intact. Thus, if the background character of the original window was set to something other than a blank, this original background could be preserved.</p> <p>The following example shows how to use overwrite() to implement a pop-up dialog box.</p> <pre>#include <curses.h> /* * Pop-up a window on top of curscr. If row and/or col * are -1 then that dimension will be centered within * curscr. Return 0 for success or -1 if malloc() failed. * Pass back the working window and the saved window for the * pop-up. The saved window should not be modified. */ int popup(work, save, nrows, ncols, row, col) WINDOW **work, **save; int nrows, ncols, row, col; { int mr, mc; getmaxyx(curscr, mr, mc); /* Windows are limited to the size of curscr. */ if (mr < nrows) nrows = mr; if (mc < ncols) ncols = mc; /* Center dimensions. */ if (row == -1) row = (mr-nrows)/2; if (col == -1) col = (mc-ncols)/2; /* The window must fit entirely in curscr. */ if (mr < row+nrows)</pre> |

```

        row = 0;
    if (mc < col+ncols)
        col = 0;
    *work = newwin(nrows, ncols, row, col);
    if (*work == NULL)
        return (-1);
    if ((*save = dupwin(*work)) == NULL) {
        delwin(*work);
        return (-1);
    }
    overwrite(curscr, *save);
    return (0);
}
/*
 * Restore the region covered by a pop-up window.
 * Delete the working window and the saved window.
 * This function is the complement to popup(). Return
 * 0 for success or -1 for an error.
 */
int
popdown(work, save)
WINDOW *work, *save;
{
    (void) wnoutrefresh(save);
    (void) delwin(save);
    (void) delwin(work);
    return (0);
}
/*
 * Compute the size of a dialog box that would fit around
 * the string.
 */
void
dialsize(str, nrows, ncols)
char *str;
int *nrows, *ncols;
{
    int rows, cols, col;
    for (rows = 1, cols = col = 0; *str != '\\0'; ++str) {
        if (*str == '\\
') {
            if (cols < col)
                cols = col;
            col = 0;
            ++rows;
        } else {
            ++col;
        }
    }
    if (cols < col)
        cols = col;
    *nrows = rows;
    *ncols = cols;
}
/*
 * Write a string into a dialog box.
 */

```

```

void
dialfill(w, s)
WINDOW *w;
char *s;
{
    int row;
    (void) wmove(w, 1, 1);
    for (row = 1; *s != '\\0'; ++s) {
        (void) waddch(w, *((unsigned char*) s));
        if (*s == '\\
')
            wmove(w, ++row, 1);
    }
    box(w, 0, 0);
}
void
dialog(str)
char *str;
{
    WINDOW *work, *save;
    int nrows, ncols, row, col;
    /* Figure out size of window. */
    dialsize(str, &nrows, &ncols);
    /* Create a centered working window with extra */
    /* room for a border. */
    (void) popup(&work, &save, nrows+2, ncols+2, -1, -1);
    /* Write text into the working window. */
    dialfill(work, str);
    /* Pause. Remember that wgetch() will do a wrefresh() */
    /* for us. */
    (void) wgetch(work);
    /* Restore curscr and free windows. */
    (void) popdown(work, save);
    /* Redraw curscr to remove window from physical screen. */
    (void) douupdate();
}

```

RETURN VALUES On success, these functions return OK . Otherwise, they return ERR .

ERRORS None.

SEE ALSO [copywin\(3XC\)](#)

| | |
|----------------------|---|
| NAME | p2open, p2close – open, close pipes to and from a command |
| SYNOPSIS | <pre>cc [flag ...] file ... -lgen [library ...] #include <libgen.h> int p2open(const char * cmd, FILE * fp [2]); int p2close(FILE * fp [2]);</pre> |
| DESCRIPTION | <p>p2open() forks and execs a shell running the command line pointed to by <i>cmd</i>. On return, <i>fp</i>[0] points to a FILE pointer to write the command's standard input and <i>fp</i>[1] points to a FILE pointer to read from the command's standard output. In this way the program has control over the input and output of the command.</p> <p>The function returns 0 if successful; otherwise, it returns -1.</p> <p>p2close() is used to close the file pointers that p2open() opened. It waits for the process to terminate and returns the process status. It returns 0 if successful; otherwise, it returns -1.</p> |
| RETURN VALUES | A common problem is having too few file descriptors. p2close() returns -1 if the two file pointers are not from the same p2open() . |
| EXAMPLES | <p>EXAMPLE 1 Example of file descriptors.</p> <pre>#include <stdio.h> #include <libgen.h> main(argc, argv) int argc; char **argv; { \011FILE *fp[2]; \011pid_t pid; \011char buf[16]; pid=p2open("/usr/bin/cat", fp); if (pid == -1) { \011\011fprintf(stderr, "p2open failed\ "); \011\011exit(1); \011}</pre> |

```

\011write(fileno(fp[0]),"This is a test\
", 16);
\011if(read(fileno(fp[1]), buf, 16) <=0)
\011\011fprintf(stderr, "p2open failed\
");
\011else
\011\011write(1, buf, 16);
\011(void)p2close(fp);
}

```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

fclose(3S) , **popen(3S)** , **setbuf(3S)** , **attributes(5)**

NOTES

Buffered writes on `fp[0]` can make it appear that the command is not listening. Judiciously placed **fflush()** calls or unbuffering `fp[0]` can be a big help; see **fclose(3S)** .

Many commands use buffered output when connected to a pipe. That, too, can make it appear as if things are not working.

Usage is not the same as for **popen()** , although it is closely related.

| | |
|---------------------------|---|
| NAME | pam – PAM (Pluggable Authentication Module) |
| SYNOPSIS | <pre>#include <security/pam_appl.h> cc [flag..] file ... -lpam [library ...]</pre> |
| DESCRIPTION | <p>The PAM framework, <code>libpam</code>, consists of an interface library and multiple authentication service modules. The PAM interface library is the layer implementing the Application Programming Interface (API). The authentication service modules are a set of dynamically loadable objects invoked by the PAM API to provide a particular type of user authentication. PAM gives system administrators the flexibility of choosing any authentication service available on the system to perform authentication. This framework also allows new authentication service modules to be plugged in and made available without modifying the applications.</p> |
| Interface Overview | <p>The PAM library interface consists of six categories of functions, the names for which all start with the prefix <code>pam_</code>.</p> <p>The first category contains functions for establishing and terminating an authentication activity, which are <code>pam_start(3)</code> and <code>pam_end(3)</code>. The functions <code>pam_set_data(3)</code> and <code>pam_get_data(3)</code> maintain module specific data. The functions <code>pam_set_item(3)</code> and <code>pam_get_item(3)</code> maintain state information. <code>pam_strerror(3)</code> is the function that returns error status information.</p> <p>The second category contains the functions that authenticate an individual user and set the credentials of the user, <code>pam_authenticate(3)</code> and <code>pam_setcred(3)</code>.</p> <p>The third category of PAM interfaces is account management. The function <code>pam_acct_mgmt(3)</code> checks for password aging and access-hour restrictions.</p> <p>Category four contains the functions that perform session management after access to the system has been granted. See <code>pam_open_session(3)</code> and <code>pam_close_session(3)</code>.</p> <p>The fifth category consists of the function that changes authentication tokens, <code>pam_chauthtok(3)</code>. An authentication token is the object used to verify the identity of the user. In UNIX, an authentication token is a user's password.</p> <p>The sixth category of functions can be used to set values for PAM environment variables. See <code>pam_putenv(3)</code>, <code>pam_getenv(3)</code>, and <code>pam_getenvlist(3)</code>.</p> <p>The <code>pam_*()</code> interfaces are implemented through the library <code>libpam</code>. For each of the categories listed above, excluding categories one and six, dynamically loadable shared modules exist that provides the appropriate service layer functionality upon demand. The functional entry points in the service layer start with the <code>pam_sm_</code> prefix. The only difference between the</p> |

Stateful Interface

`pam_sm_*`() interfaces and their corresponding `pam_` interfaces is that all the `pam_sm_*`() interfaces require extra parameters to pass service-specific options to the shared modules. Refer to `pam_sm(3)` for an overview of the PAM service module APIs.

A sequence of calls sharing a common set of state information is referred to as an authentication transaction. An authentication transaction begins with a call to `pam_start()`. `pam_start()` allocates space, performs various initialization activities, and assigns a PAM authentication handle to be used for subsequent calls to the library.

After initiating an authentication transaction, applications can invoke `pam_authenticate()` to authenticate a particular user, and `pam_acct_mgmt()` to perform system entry management. For example, the application may want to determine if the user's password has expired.

If the user has been successfully authenticated, the application calls `pam_setcred()` to set any user credentials associated with the authentication service. Within one authentication transaction (between `pam_start()` and `pam_end()`), all calls to the PAM interface should be made with the same authentication handle returned by `pam_start()`. This is necessary because certain service modules may store module-specific data in a handle that is intended for use by other modules. For example, during the call to `pam_authenticate()`, service modules may store data in the handle that is intended for use by `pam_setcred()`.

To perform session management, applications call `pam_open_session()`. Specifically, the system may want to store the total time for the session. The function `pam_close_session()` closes the current session.

When necessary, applications can call `pam_get_item()` and `pam_set_item()` to access and to update specific authentication information. Such information may include the current username.

To terminate an authentication transaction, the application simply calls `pam_end()`, which frees previously allocated space used to store authentication information.

**Application-
Authentication
Service Interactive
Interface**

The authentication service in PAM does not communicate directly with the user; instead it relies on the application to perform all such interactions. The application passes a pointer to the function, `conv()`, along with any associated application data pointers, through a `pam_conv` structure to the authentication service when it initiates an authentication transaction, via a call to `pam_start()`. The service will then use the function, `conv()`, to prompt the user for data, output error messages, and display text information. Refer to `pam_start(3)` for more information.

Stacking Multiple Schemes

The PAM architecture enables authentication by multiple authentication services through *stacking*. System entry applications, such as `login(1)`, stack multiple service modules to authenticate users with multiple authentication services. The order in which authentication service modules are stacked is specified in the configuration file, `pam.conf(4)`. A system administrator determines this ordering, and also determines whether the same password can be used for all authentication services.

Administrative Interface

The authentication library, `/usr/lib/libpam.so.1`, implements the framework interface. Various authentication services are implemented by their own loadable modules whose paths are specified through the `pam.conf(4)` file.

RETURN VALUES

The PAM functions may return one of the following generic values, or one of the values defined in the specific man pages:

| | |
|------------------------------|---|
| <code>PAM_SUCCESS</code> | The function returned successfully. |
| <code>PAM_OPEN_ERR</code> | dlopen() failed when dynamically loading a service module. |
| <code>PAM_SYMBOL_ERR</code> | Symbol not found. |
| <code>PAM_SERVICE_ERR</code> | Error in service module. |
| <code>PAM_SYSTEM_ERR</code> | System error. |
| <code>PAM_BUF_ERR</code> | Memory buffer error. |
| <code>PAM_CONV_ERR</code> | Conversation failure. |
| <code>PAM_PERM_DENIED</code> | Permission denied. |

ATTRIBUTES

See `attributes(5)` for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT Level | MT-Safe with exceptions |

SEE ALSO

`login(1)`, `pam_authenticate(3)`, `pam_chauthtok(3)`, `pam_open_session(3)`, `pam_set_item(3)`, `pam_setcred(3)`, `pam_sm(3)`, `pam_start(3)`, `pam_strerror(3)`, `pam.conf(4)`, `attributes(5)`

WARNING

Please note that all the PAM APIs and their data structures are subject to change without notice.

NOTES

The interfaces in **libpam()** are MT-safe only if each thread within the multi-threaded application uses its own PAM handle.

| NAME | pam_acct_mgmt – perform PAM account validation procedures | | | | | | | | |
|--------------------------|---|------------------|--|--------------------------|---|---------------------|---|------------------|---------------------------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h></pre> <pre>int pam_acct_mgmt(pam_handle_t *pamh, int flags);</pre> | | | | | | | | |
| DESCRIPTION | <p>The function pam_acct_mgmt() is called to determine if the current user's account is valid. It checks for password and account expiration, and verifies access hour restrictions. This function is typically called after the user has been authenticated with pam_authenticate(3).</p> <p>The <i>pamh</i> argument is an authentication handle obtained by a prior call to pam_start(). The following flags may be set in the <i>flags</i> field:</p> <table border="0"> <tr> <td>PAM_SILENT</td> <td>The account management service should not generate any messages.</td> </tr> <tr> <td>PAM_DISALLOW_NULL_AUTHOK</td> <td>The account management service should return PAM_NEW_AUTHOK_REQD if the user has a null authentication token.</td> </tr> </table> | PAM_SILENT | The account management service should not generate any messages. | PAM_DISALLOW_NULL_AUTHOK | The account management service should return PAM_NEW_AUTHOK_REQD if the user has a null authentication token. | | | | |
| PAM_SILENT | The account management service should not generate any messages. | | | | | | | | |
| PAM_DISALLOW_NULL_AUTHOK | The account management service should return PAM_NEW_AUTHOK_REQD if the user has a null authentication token. | | | | | | | | |
| RETURN VALUES | <p>Upon successful completion, PAM_SUCCESS is returned. In addition to the error return values described in pam(3), the following values may be returned:</p> <table border="0"> <tr> <td>PAM_USER_UNKNOWN</td> <td>User not known to underlying account management module.</td> </tr> <tr> <td>PAM_AUTH_ERR</td> <td>Authentication failure.</td> </tr> <tr> <td>PAM_NEW_AUTHOK_REQD</td> <td>New authentication token required. This is normally returned if the machine security policies require that the password should be changed because the password is NULL or has aged.</td> </tr> <tr> <td>PAM_ACCT_EXPIRED</td> <td>User account has expired.</td> </tr> </table> | PAM_USER_UNKNOWN | User not known to underlying account management module. | PAM_AUTH_ERR | Authentication failure. | PAM_NEW_AUTHOK_REQD | New authentication token required. This is normally returned if the machine security policies require that the password should be changed because the password is NULL or has aged. | PAM_ACCT_EXPIRED | User account has expired. |
| PAM_USER_UNKNOWN | User not known to underlying account management module. | | | | | | | | |
| PAM_AUTH_ERR | Authentication failure. | | | | | | | | |
| PAM_NEW_AUTHOK_REQD | New authentication token required. This is normally returned if the machine security policies require that the password should be changed because the password is NULL or has aged. | | | | | | | | |
| PAM_ACCT_EXPIRED | User account has expired. | | | | | | | | |
| ATTRIBUTES | <p>See attributes(5) for description of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT Level</td> <td>MT-Safe with exceptions</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT Level | MT-Safe with exceptions | | | | |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | | | |
| MT Level | MT-Safe with exceptions | | | | | | | | |

SEE ALSO | pam(3), pam_authenticate(3), pam_start(3), libpam(4), attributes(5)

NOTES | The interfaces in **libpam()** are MT-safe only if each thread within the multi-threaded application uses its own PAM handle.

| | | | | | | | | | | | |
|---------------------------|---|--------------|--|---------------------------|--|----------------------|---|------------------|---|--------------|--|
| NAME | pam_authenticate – perform authentication within the PAM framework | | | | | | | | | | |
| SYNOPSIS | <pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h></pre> <pre>int pam_authenticate(pam_handle_t *pamh, int flags);</pre> | | | | | | | | | | |
| DESCRIPTION | <p>pam_authenticate() is called to authenticate the current user. The user is usually required to enter a password or similar authentication token depending upon the authentication service configured within the system. The user in question should have been specified by a prior call to pam_start() or pam_set_item().</p> <p>The following flags may be set in the <i>flags</i> field:</p> <table border="0"> <tr> <td>PAM_SILENT</td> <td>Authentication service should not generate any messages.</td> </tr> <tr> <td>PAM_DISALLOW_NULL_AUTHTOK</td> <td>The authentication service should return PAM_AUTH_ERROR if the user has a null authentication token.</td> </tr> </table> | PAM_SILENT | Authentication service should not generate any messages. | PAM_DISALLOW_NULL_AUTHTOK | The authentication service should return PAM_AUTH_ERROR if the user has a null authentication token. | | | | | | |
| PAM_SILENT | Authentication service should not generate any messages. | | | | | | | | | | |
| PAM_DISALLOW_NULL_AUTHTOK | The authentication service should return PAM_AUTH_ERROR if the user has a null authentication token. | | | | | | | | | | |
| RETURN VALUES | <p>Upon successful completion, PAM_SUCCESS is returned. In addition to the error return values described in pam(3), the following values may be returned:</p> <table border="0"> <tr> <td>PAM_AUTH_ERR</td> <td>Authentication failure.</td> </tr> <tr> <td>PAM_CRED_INSUFFICIENT</td> <td>Cannot access authentication data due to insufficient credentials.</td> </tr> <tr> <td>PAM_AUTHINFO_UNAVAIL</td> <td>Underlying authentication service cannot retrieve authentication information.</td> </tr> <tr> <td>PAM_USER_UNKNOWN</td> <td>User not known to the underlying authentication module.</td> </tr> <tr> <td>PAM_MAXTRIES</td> <td>An authentication service has maintained a retry count which has been reached. No further retries should be attempted.</td> </tr> </table> | PAM_AUTH_ERR | Authentication failure. | PAM_CRED_INSUFFICIENT | Cannot access authentication data due to insufficient credentials. | PAM_AUTHINFO_UNAVAIL | Underlying authentication service cannot retrieve authentication information. | PAM_USER_UNKNOWN | User not known to the underlying authentication module. | PAM_MAXTRIES | An authentication service has maintained a retry count which has been reached. No further retries should be attempted. |
| PAM_AUTH_ERR | Authentication failure. | | | | | | | | | | |
| PAM_CRED_INSUFFICIENT | Cannot access authentication data due to insufficient credentials. | | | | | | | | | | |
| PAM_AUTHINFO_UNAVAIL | Underlying authentication service cannot retrieve authentication information. | | | | | | | | | | |
| PAM_USER_UNKNOWN | User not known to the underlying authentication module. | | | | | | | | | | |
| PAM_MAXTRIES | An authentication service has maintained a retry count which has been reached. No further retries should be attempted. | | | | | | | | | | |
| ATTRIBUTES | See attributes(5) for description of the following attributes: | | | | | | | | | | |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT Level | MT-Safe with exceptions |

SEE ALSO

pam(3), **pam_open_session(3)**, **pam_set_item(3)**, **pam_setcred(3)**, **pam_start(3)**, **libpam(4)**, **attributes(5)**

NOTES

In the case of authentication failures due to an incorrect username or password, it is the responsibility of the application to retry **pam_authenticate()** and to maintain the retry count. An authentication service module may implement an internal retry count and return an error `PAM_MAXTRIES` if the module does not want the application to retry.

If the PAM framework cannot load the authentication module, then it will return `PAM_ABORT`. This indicates a serious failure, and the application should not attempt to retry the authentication.

For security reasons, the location of authentication failures is hidden from the user. Thus, if several authentication services are stacked and a single service fails, **pam_authenticate()** requires that the user re-authenticate each of the services.

A null authentication token in the authentication database will result in successful authentication unless `PAM_DISALLOW_NULL_AUTHOK` was specified. In such cases, there will be no prompt to the user to enter an authentication token.

The interfaces in **libpam()** are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

| | | | | | | | | | | | | | | | |
|-----------------------------|---|-----------------|--|-----------------------------|--|---------------------------|---|------------------------|---------------------------------|----------------------------|--------------------------------------|------------------|-----------------------------------|---------------|---|
| NAME | pam_chauthtok – perform password related functions within the PAM framework | | | | | | | | | | | | | | |
| SYNOPSIS | <pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h></pre> <pre>int pam_chauthtok(pam_handle_t *pamh, const int flags);</pre> | | | | | | | | | | | | | | |
| DESCRIPTION | <p>pam_chauthtok() is called to change the authentication token associated with a particular user referenced by the authentication handle, <i>pamh</i>.</p> <p>The following flag may be passed in to pam_chauthtok():</p> <table border="0"> <tr> <td style="padding-right: 20px;">PAM_SILENT</td> <td>The password service should not generate any messages.</td> </tr> <tr> <td style="padding-right: 20px;">PAM_CHANGE_EXPIRED_AUTH Tok</td> <td>The password service should only update those passwords that have aged. If this flag is not passed, all password services should update their passwords.</td> </tr> </table> <p>Upon successful completion of the call, the authentication token of the user will be changed in accordance with the password service configured in the system through pam.conf(4).</p> | PAM_SILENT | The password service should not generate any messages. | PAM_CHANGE_EXPIRED_AUTH Tok | The password service should only update those passwords that have aged. If this flag is not passed, all password services should update their passwords. | | | | | | | | | | |
| PAM_SILENT | The password service should not generate any messages. | | | | | | | | | | | | | | |
| PAM_CHANGE_EXPIRED_AUTH Tok | The password service should only update those passwords that have aged. If this flag is not passed, all password services should update their passwords. | | | | | | | | | | | | | | |
| RETURN VALUES | <p>Upon successful completion, PAM_SUCCESS is returned. In addition to the error return values described in pam(3), the following values may be returned:</p> <table border="0"> <tr> <td style="padding-right: 20px;">PAM_PERM_DENIED</td> <td>No permission.</td> </tr> <tr> <td style="padding-right: 20px;">PAM_AUTH Tok_ERR</td> <td>Authentication token manipulation error.</td> </tr> <tr> <td style="padding-right: 20px;">PAM_AUTH Tok_RECOVERY_ERR</td> <td>Authentication information cannot be recovered.</td> </tr> <tr> <td style="padding-right: 20px;">PAM_AUTH Tok_LOCK_BUSY</td> <td>Authentication token lock busy.</td> </tr> <tr> <td style="padding-right: 20px;">PAM_AUTH Tok_DISABLE_AGING</td> <td>Authentication token aging disabled.</td> </tr> <tr> <td style="padding-right: 20px;">PAM_USER_UNKNOWN</td> <td>User unknown to password service.</td> </tr> <tr> <td style="padding-right: 20px;">PAM_TRY_AGAIN</td> <td>Preliminary check by password service failed.</td> </tr> </table> | PAM_PERM_DENIED | No permission. | PAM_AUTH Tok_ERR | Authentication token manipulation error. | PAM_AUTH Tok_RECOVERY_ERR | Authentication information cannot be recovered. | PAM_AUTH Tok_LOCK_BUSY | Authentication token lock busy. | PAM_AUTH Tok_DISABLE_AGING | Authentication token aging disabled. | PAM_USER_UNKNOWN | User unknown to password service. | PAM_TRY_AGAIN | Preliminary check by password service failed. |
| PAM_PERM_DENIED | No permission. | | | | | | | | | | | | | | |
| PAM_AUTH Tok_ERR | Authentication token manipulation error. | | | | | | | | | | | | | | |
| PAM_AUTH Tok_RECOVERY_ERR | Authentication information cannot be recovered. | | | | | | | | | | | | | | |
| PAM_AUTH Tok_LOCK_BUSY | Authentication token lock busy. | | | | | | | | | | | | | | |
| PAM_AUTH Tok_DISABLE_AGING | Authentication token aging disabled. | | | | | | | | | | | | | | |
| PAM_USER_UNKNOWN | User unknown to password service. | | | | | | | | | | | | | | |
| PAM_TRY_AGAIN | Preliminary check by password service failed. | | | | | | | | | | | | | | |
| ATTRIBUTES | See attributes(5) for description of the following attributes: | | | | | | | | | | | | | | |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT Level | MT-Safe with exceptions |

SEE ALSO `login(1)`, `passwd(1)`, `pam(3)`, `pam_authenticate(3)`, `pam_start(3)`,
attributes

NOTES The flag `PAM_CHANGE_EXPIRED_AUTHTOK` is typically used by a `login` application which has determined that the user's password has aged or expired. Before allowing the user to login, the `login` application may invoke `pam_chauthtok()` with this flag to allow the user to update the password. Typically applications such as `passwd(1)` should not use this flag.

`pam_chauthtok()` performs a preliminary check before attempting to update passwords. This check is performed for each password module in the stack as listed in `pam.conf(4)`. The check may include pinging remote name services to determine if they are available. If `pam_chauthtok()` returns `PAM_TRY_AGAIN`, then the check has failed, and passwords are not updated.

The interfaces in `libpam()` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

NAME pam_getenv - returns the value for a PAM environment name

SYNOPSIS `cc [flag ...] file ... -lpam [library ...]
#include <security/pam_appl.h>`

```
char *pam_getenv(pam_handle_t *pamh, const char *name);
```

DESCRIPTION **pam_getenv()** searches the PAM handle, *pamh*, for a value associated with *name*. If a value is present, **pam_getenv()** makes a copy of the value and returns a pointer to the copy back to the calling application. If no such entry exists, **pam_getenv()** returns NULL. It is the responsibility of the calling application to free the memory returned by **pam_getenv()**.

RETURN VALUES If successful, **pam_getenv()** returns a copy of the *value* associated with *name* in the PAM handle; otherwise, it returns a NULL pointer.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT Level | MT-Safe with exceptions |

SEE ALSO **pam(3)**, **pam_getenvlist(3)**, **pam_putenv(3)**, **libpam(4)**, **attributes(5)**

NOTES The interfaces in **libpam()** are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

| NAME | pam_getenvlist – returns a list of all the PAM environment variables | | | | |
|----------------------|---|----------------|-----------------|----------|-------------------------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> char **pam_getenvlist(pam_handle_t *pamh);</pre> | | | | |
| DESCRIPTION | <p>pam_getenvlist() returns a list of all the PAM environment variables stored in the PAM handle, <i>pamh</i>. The list is returned as a null-terminated array of pointers to strings. Each string contains a single PAM environment variable of the form <i>name=value</i>. The list returned is a duplicate copy of all the environment variables stored in <i>pamh</i>. It is the responsibility of the calling application to free the memory returned by pam_getenvlist().</p> | | | | |
| RETURN VALUES | <p>If successful pam_getenvlist() returns, in a null-terminated array, a copy of all the PAM environment variables stored in <i>pamh</i>. Upon error, pam_getenvlist() returns a null pointer.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" data-bbox="500 989 1399 1077"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT Level</td> <td>MT-Safe with exceptions</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT Level | MT-Safe with exceptions |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT Level | MT-Safe with exceptions | | | | |
| SEE ALSO | pam(3) , pam_getenv(3) , pam_putenv(3) , libpam(4) , attributes(5) | | | | |
| NOTES | <p>The interfaces in libpam() are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.</p> | | | | |

NAME pam_get_user – PAM routine to retrieve user name

SYNOPSIS `cc [flag ...] file ... -lpam [library ...]
#include <security/pam_appl.h>`

```
int pam_get_user(pam_handle_t *pamh, char **user, const char *prompt);
```

DESCRIPTION

pam_get_user() is used by PAM service modules to retrieve the current user name from the PAM handle. If the user name has not been set via **pam_start()** or **pam_set_item()**, then the PAM conversation function will be used to prompt the user for the user name with the string "prompt". If *prompt* is NULL, then **pam_get_item()** is called and the value of PAM_USER_PROMPT is used for prompting. If the value of PAM_USER_PROMPT is NULL, the following default prompt is used:

```
Please enter user name:
```

After the user name is gathered by the conversation function, **pam_set_item()** is called to set the value of PAM_USER. By convention, applications that need to prompt for a user name should call **pam_set_item()** and set the value of PAM_USER_PROMPT before calling **pam_authenticate()**. The service module's **pam_sm_authenticate()** function will then call **pam_get_user()** to prompt for the user name.

Note that certain PAM service modules, such as a smart card module, may override the value of PAM_USER_PROMPT and pass in their own prompt. Applications that call **pam_authenticate()** multiple times should set the value of PAM_USER to NULL with **pam_set_item()** before calling **pam_authenticate()**, if they want the user to be prompted for a new user name each time. The value of *user* retrieved by **pam_get_user()** should not be modified or freed. The item will be released by **pam_end()**.

RETURN VALUES

Upon success **pam_get_user()** returns PAM_SUCCESS; otherwise it returns an error code. Refer to **pam(3)** for information on error related return values.

ATTRIBUTES

See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT Level | MT-Safe with exceptions |

SEE ALSO

pam(3), pam_authenticate(3), pam_end(3), pam_get_item(3),
pam_set_item(3), pam_sm(3), pam_sm_authenticate(3), pam_start(3),
attributes(5)

NOTES

The interfaces in **libpam()** are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

NAME pam_open_session, pam_close_session – perform PAM session creation and termination operations

SYNOPSIS

```
cc
[
  flag
  ... ]
file
...
-lpam
[
  library
  ... ]
#include <security/pam_appl.h>
```

```
int pam_open_session(pam_handle_t * pamh, int flags);
```

```
int pam_close_session(pam_handle_t * pamh, int flags);
```

DESCRIPTION

pam_open_session() is called after a user has been successfully authenticated. See **pam_authenticate(3)** and **pam_acct_mgmt(3)**. It is used to notify the session modules that a new session has been initiated. All programs that use the **pam(3)** library should invoke **pam_open_session()** when beginning a new session. Upon termination of this activity, **pam_close_session()** should be invoked to inform **pam(3)** that the session has terminated.

The *pamh* argument is an authentication handle obtained by a prior call to **pam_start()**. The following flag may be set in the *flags* field for **pam_open_session()** and **pam_close_session()**:

PAM_SILENT The session service should not generate any messages.

RETURN VALUES

Upon successful completion, **PAM_SUCCESS** is returned. In addition to the return values defined in **pam(3)**, the following value may be returned on error:

PAM_SESSION_ERR Cannot make or remove an entry for the specified session.

ATTRIBUTES

See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT Level | MT-Safe with exceptions |

SEE ALSO | `getutxent(3C)`, `pam(3)`, `pam_acct_mgmt(3)`, `pam_authenticate(3)`,
`pam_start(3)`, `attributes(5)`

NOTES | In many instances, the `pam_open_session()` and `pam_close_session()` calls may be made by different processes. For example, in UNIX the `login` process opens a session, while the `init` process closes the session. In this case, UTMP/WTMP entries may be used to link the call to `pam_close_session()` with an earlier call to `pam_open_session()`. This is possible because UTMP/WTMP entries are uniquely identified by a combination of attributes, including the user login name and device name, which are accessible through the PAM handle, `pamh`. The call to `pam_open_session()` should precede UTMP/WTMP entry management, and the call to `pam_close_session()` should follow UTMP/WTMP exit management.

The interfaces in `libpam()` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

| | | | | | | | | | | | | | | | | | |
|----------------------|---|-------------|-------------------------------------|--------------|---|----------------|-------------------|-----------------|--------------------------|----------------|---------------|-------------|----------------------|--------------|-----------------------|-----------------|--------------------|
| NAME | pam_putenv – change or add a value to the PAM environment | | | | | | | | | | | | | | | | |
| SYNOPSIS | <pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h></pre> <pre>int pam_putenv(pam_handle_t *pamh, const char *name_value);</pre> | | | | | | | | | | | | | | | | |
| DESCRIPTION | <p>The pam_putenv() routine sets the value of the PAM environment variable <i>name</i> equal to <i>value</i> either by altering an existing PAM variable or by creating a new one.</p> <p>The variable <i>name_value</i> points to a string of the form <i>name=value</i>. A call to pam_putenv() does not immediately change the environment. All <i>name_value</i> pairs are stored in the PAM handle, <i>pamh</i>. An application such as login(1) may make a call to pam_getenv(3) or pam_getenvlist(3) to retrieve the PAM environment variables saved in the PAM handle and set them in the environment if appropriate. login will not set PAM environment values which overwrite the values for SHELL, HOME, LOGNAME, MAIL, CDPATH, IFS, and PATH. Nor will login set PAM environment values which overwrite any value that begins with LD_.</p> <p>If <i>name_value</i> equals NAME=, then the value associated with NAME in the PAM handle will be set to an empty value. If <i>name_value</i> equals NAME, then the environment variable NAME will be removed from the PAM handle.</p> | | | | | | | | | | | | | | | | |
| RETURN VALUES | <p>The function pam_putenv() may return one of the following values:</p> <table border="0"> <tr> <td style="padding-right: 20px;">PAM_SUCCESS</td> <td>The function returned successfully.</td> </tr> <tr> <td>PAM_OPEN_ERR</td> <td>dlopen() failed when dynamically loading a service module.</td> </tr> <tr> <td>PAM_SYMBOL_ERR</td> <td>Symbol not found.</td> </tr> <tr> <td>PAM_SERVICE_ERR</td> <td>Error in service module.</td> </tr> <tr> <td>PAM_SYSTEM_ERR</td> <td>System error.</td> </tr> <tr> <td>PAM_BUF_ERR</td> <td>Memory buffer error.</td> </tr> <tr> <td>PAM_CONV_ERR</td> <td>Conversation failure.</td> </tr> <tr> <td>PAM_PERM_DENIED</td> <td>Permission denied.</td> </tr> </table> | PAM_SUCCESS | The function returned successfully. | PAM_OPEN_ERR | dlopen() failed when dynamically loading a service module. | PAM_SYMBOL_ERR | Symbol not found. | PAM_SERVICE_ERR | Error in service module. | PAM_SYSTEM_ERR | System error. | PAM_BUF_ERR | Memory buffer error. | PAM_CONV_ERR | Conversation failure. | PAM_PERM_DENIED | Permission denied. |
| PAM_SUCCESS | The function returned successfully. | | | | | | | | | | | | | | | | |
| PAM_OPEN_ERR | dlopen() failed when dynamically loading a service module. | | | | | | | | | | | | | | | | |
| PAM_SYMBOL_ERR | Symbol not found. | | | | | | | | | | | | | | | | |
| PAM_SERVICE_ERR | Error in service module. | | | | | | | | | | | | | | | | |
| PAM_SYSTEM_ERR | System error. | | | | | | | | | | | | | | | | |
| PAM_BUF_ERR | Memory buffer error. | | | | | | | | | | | | | | | | |
| PAM_CONV_ERR | Conversation failure. | | | | | | | | | | | | | | | | |
| PAM_PERM_DENIED | Permission denied. | | | | | | | | | | | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | | | | | | | | | | | | | |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT Level | MT-Safe with exceptions |

SEE ALSO `dlopen(3X)`, `pam(3)`, `pam_getenv(3)`, `pam_getenvlist(3)`, `libpam(4)`, `attributes(5)`

NOTES The interfaces in `libpam()` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

| | | | | | | | | | | | |
|-----------------------|--|--------------------|--|------------------|--|-----------------------|--|------------------|--------------------------------------|------------|--|
| NAME | pam_setcred – modify/delete user credentials for an authentication service | | | | | | | | | | |
| SYNOPSIS | <pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h></pre> <p>int pam_setcred(pam_handle_t *pamh, int flags);</p> | | | | | | | | | | |
| DESCRIPTION | <p>pam_setcred() is used to establish, modify, or delete user credentials. pam_setcred() is typically called after the user has been authenticated and after a session has been opened. See pam_authenticate(3), pam_acct_mgmt(3), and pam_open_session(3).</p> <p>The user is specified by a prior call to pam_start() or pam_set_item(), and is referenced by the authentication handle, <i>pamh</i>. The following flags may be set in the <i>flags</i> field. Note that the first four flags are mutually exclusive:</p> <table border="0"> <tr> <td>PAM_ESTABLISH_CRED</td> <td>Set user credentials for an authentication service.</td> </tr> <tr> <td>PAM_DELETE_CRED</td> <td>Delete user credentials associated with an authentication service.</td> </tr> <tr> <td>PAM_REINITIALIZE_CRED</td> <td>Reinitialize user credentials.</td> </tr> <tr> <td>PAM_REFRESH_CRED</td> <td>Extend lifetime of user credentials.</td> </tr> <tr> <td>PAM_SILENT</td> <td>Authentication service should not generate any messages.</td> </tr> </table> <p>If no flag is set, PAM_ESTABLISH_CRED is used as the default.</p> | PAM_ESTABLISH_CRED | Set user credentials for an authentication service. | PAM_DELETE_CRED | Delete user credentials associated with an authentication service. | PAM_REINITIALIZE_CRED | Reinitialize user credentials. | PAM_REFRESH_CRED | Extend lifetime of user credentials. | PAM_SILENT | Authentication service should not generate any messages. |
| PAM_ESTABLISH_CRED | Set user credentials for an authentication service. | | | | | | | | | | |
| PAM_DELETE_CRED | Delete user credentials associated with an authentication service. | | | | | | | | | | |
| PAM_REINITIALIZE_CRED | Reinitialize user credentials. | | | | | | | | | | |
| PAM_REFRESH_CRED | Extend lifetime of user credentials. | | | | | | | | | | |
| PAM_SILENT | Authentication service should not generate any messages. | | | | | | | | | | |
| RETURN VALUES | <p>Upon success, pam_setcred() returns PAM_SUCCESS. In addition to the error return values described in pam(3) the following values may be returned upon error:</p> <table border="0"> <tr> <td>PAM_CRED_UNAVAIL</td> <td>Underlying authentication service can not retrieve user credentials unavailable.</td> </tr> <tr> <td>PAM_CRED_EXPIRED</td> <td>User credentials expired.</td> </tr> <tr> <td>PAM_USER_UNKNOWN</td> <td>User unknown to underlying authentication service.</td> </tr> <tr> <td>PAM_CRED_ERR</td> <td>Failure setting user credentials.</td> </tr> </table> | PAM_CRED_UNAVAIL | Underlying authentication service can not retrieve user credentials unavailable. | PAM_CRED_EXPIRED | User credentials expired. | PAM_USER_UNKNOWN | User unknown to underlying authentication service. | PAM_CRED_ERR | Failure setting user credentials. | | |
| PAM_CRED_UNAVAIL | Underlying authentication service can not retrieve user credentials unavailable. | | | | | | | | | | |
| PAM_CRED_EXPIRED | User credentials expired. | | | | | | | | | | |
| PAM_USER_UNKNOWN | User unknown to underlying authentication service. | | | | | | | | | | |
| PAM_CRED_ERR | Failure setting user credentials. | | | | | | | | | | |
| ATTRIBUTES | See attributes(5) for description of the following attributes: | | | | | | | | | | |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT Level | MT-Safe with exceptions |

SEE ALSO

`pam(3)`, `pam_acct_mgmt(3)`, `pam_authenticate(3)`,
`pam_open_session(3)`, `pam_set_item(3)`, `pam_start(3)`, `libpam(4)`,
`attributes(5)`

NOTES

The interfaces in `libpam()` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

| | |
|--------------------|--|
| NAME | pam_set_data, pam_get_data – PAM routines to maintain module specific state |
| SYNOPSIS | <pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> int pam_set_data(pam_handle_t * pamh, const char * module_data_name, void * data, void (* cleanup)(pam_handle_t * pamh, void * data, int pam_end_status)); int pam_get_data(const pam_handle_t * pamh, const char * module_data_name, const void ** data);</pre> |
| DESCRIPTION | <p>pam_set_data() and pam_get_data() allow PAM service modules to access and update module specific information as needed. These functions should not be used by applications.</p> <p>pam_set_data() stores module specific data within the PAM handle, <i>pamh</i> . The <i>module_data_name</i> argument uniquely identifies the data, and the <i>data</i> argument represents the actual data. <i>module_data_name</i> should be unique across all services (UNIX, etc).</p> <p>The <i>cleanup</i> function frees up any memory used by the <i>data</i> after it is no longer needed, and is invoked by pam_end() . The <i>cleanup</i> function takes as its arguments a pointer to the PAM handle, <i>pamh</i> , a pointer to the actual data, <i>data</i> , and a status code, <i>pam_end_status</i> . The status code determines exactly what state information needs to be purged.</p> <p>If pam_set_data() is called and module data already exists from a prior call to pam_set_data() under the same <i>module_data_name</i> , then the existing <i>data</i> is replaced by the new <i>data</i> , and the existing <i>cleanup</i> function is replaced by the new <i>cleanup</i> function.</p> <p>pam_get_data() retrieves module-specific data stored in the PAM handle, <i>pamh</i> , identified by the unique name, <i>module_data_name</i> . The <i>data</i> argument is assigned the address of the requested data. The <i>data</i> retrieved by pam_get_data() should not be modified or freed. The <i>data</i> will be released by pam_end() .</p> |

RETURN VALUES

In addition to the return values listed in `pam(3)`, the following value may also be returned:

`PAM_NO_MODULE_DATA` No module specific data is present.

ATTRIBUTES

See `attributes(5)` for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT Level | MT-Safe with exceptions |

SEE ALSO

`pam(3)`, `pam_end(3)`, `libpam(4)`, `attributes(5)`

NOTES

The interfaces in `libpam()` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

NAME | pam_set_item, pam_get_item – authentication information routines for PAM

SYNOPSIS |

```
cc
[
flag
... ]
file
...
-lpam
[
library
... ]
#include <security/pam_appl.h>
```

DESCRIPTION |

```
int pam_set_item(pam_handle_t * pamh, int item_type, const void * item);
int pam_get_item(const pam_handle_t * pamh, int item_type, void ** item);
```

pam_get_item() and **pam_set_item()** allow applications and PAM service modules to access and to update PAM information as needed. The information is specified by *item_type* , and can be one of the following:

| | |
|-----------------|--|
| PAM_SERVICE | The service name. |
| PAM_USER | The user name. |
| PAM_AUTHTOK | The user authentication token. |
| PAM_OLDAUTHTOK | The old user authentication token. |
| PAM_TTY | The tty name. |
| PAM_RHOST | The remote host name. |
| PAM_RUSER | The remote user name. |
| PAM_CONV | The <code>pam_conv</code> structure. |
| PAM_USER_PROMPT | The default prompt used by pam_get_user() . |

For security reasons, the *item_type* PAM_AUTHTOK and PAM_OLDAUTHTOK are available only to the module providers. The authentication module, account module, and session management module should treat PAM_AUTHTOK as the current authentication token and ignore PAM_OLDAUTHTOK . The password management module should treat PAM_OLDAUTHTOK as the current authentication token and PAM_AUTHTOK as the new authentication token.

pam_set_item() is passed the authentication handle, *pamh* , returned by **pam_start()** , a pointer to the object, *item* , and its type, *item_type* . If successful,

pam_set_item() copies the item to an internal storage area allocated by the authentication module and returns `PAM_SUCCESS` . An item that had been previously set will be overwritten by the new value.

pam_get_item() is passed the authentication handle, *pamh* , returned by **pam_start()** , an *item_type* , and the address of the pointer, *item* , which is assigned the address of the requested object. The object data is valid until modified by a subsequent call to **pam_set_item()** for the same *item_type* , or unless it is modified by any of the underlying service modules. If the item has not been previously set, **pam_get_item()** returns a null pointer. An *item* retrieved by **pam_get_item()** should not be modified or freed. The item will be released by **pam_end()** .

RETURN VALUES

Upon success **pam_get_item()** returns `PAM_SUCCESS`; otherwise it returns an error code. Refer to **pam(3)** for information on error related return values.

ATTRIBUTES

See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT Level | MT-Safe with exceptions |

SEE ALSO

pam(3) , **pam_acct_mgmt(3)** , **pam_authenticate(3)** , **pam_chauthtok(3)** , **pam_get_user(3)** , **pam_open_session(3)** , **pam_setcred(3)** , **pam_start(3)** , **attributes(5)**

NOTES

The interfaces in **libpam()** are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

| | |
|---------------------------|--|
| NAME | pam_sm – PAM Service Module APIs |
| SYNOPSIS | <pre>#include <security/pam_appl.h> #include <security/pam_modules.h> cc [flag ...] file ... -lpam [library ...]</pre> |
| DESCRIPTION | <p>PAM gives system administrators the flexibility of choosing any authentication service available on the system to perform authentication. The framework also allows new authentication service modules to be plugged in and made available without modifying the applications.</p> <p>The PAM framework, <code>libpam</code>, consists of an interface library and multiple authentication service modules. The PAM interface library is the layer implementing the Application Programming Interface (API). The authentication service modules are a set of dynamically loadable objects invoked by the PAM API to provide a particular type of user authentication.</p> <p>This manual page gives an overview of the PAM APIs for the service modules.</p> |
| Interface Overview | <p>The PAM service module interface consists of functions which can be grouped into four categories. The names for all the authentication library functions start with <code>pam_sm</code>. The only difference between the <code>pam_*</code>() interfaces and their corresponding <code>pam_sm_*</code>() interfaces is that all the <code>pam_sm_*</code>() interfaces require extra parameters to pass service-specific options to the shared modules. They are otherwise identical.</p> <p>The first category contains functions to authenticate an individual user, <code>pam_sm_authenticate(3)</code>, and to set the credentials of the user, <code>pam_sm_setcred(3)</code>. These back-end functions implement the functionality of <code>pam_authenticate(3)</code> and <code>pam_setcred(3)</code> respectively.</p> <p>The second category contains the function to do account management: <code>pam_sm_acct_mgmt(3)</code>. This includes checking for password aging and access-hour restrictions. This back-end function implements the functionality of <code>pam_acct_mgmt(3)</code>.</p> <p>The third category contains the functions <code>pam_sm_open_session(3)</code> and <code>pam_sm_close_session(3)</code> to perform session management after access to the system has been granted. These back-end functions implement the functionality of <code>pam_open_session(3)</code> and <code>pam_close_session(3)</code>, respectively.</p> <p>The fourth category consists a function to change authentication tokens <code>pam_sm_chauthtok(3)</code>. This back-end function implements the functionality of <code>pam_chauthtok(3)</code>.</p> |

Stateful Interface

A sequence of calls sharing a common set of state information is referred to as an authentication transaction. An authentication transaction begins with a call to **pam_start()**. **pam_start()** allocates space, performs various initialization activities, and assigns an authentication handle to be used for subsequent calls to the library. Note that the service modules do not get called or initialized when **pam_start()** is called. The modules are loaded and the symbols resolved upon first use of that function.

The PAM handle keeps certain information about the transaction that can be accessed through the **pam_get_item()** API. Though the modules can also use **pam_set_item()** to change any of the item information, it is recommended that nothing be changed except `PAM_AUTHTOK` and `PAM_OLDAUTHTOK`.

If the modules want to store any module specific state information then they can use the **pam_set_data(3)** function to store that information with the PAM handle. The data should be stored with a name which is unique across all modules and module types. For example, `SUNW_PAM_UNIX_AUTH_userid` can be used as a name by the UNIX module to store information about the state of user's authentication. Some modules use this technique to share data across two different module types.

Also, during the call to **pam_authenticate()**, the UNIX module may store the authentication status (success or reason for failure) in the handle, using a unique name such as `SUNW_SECURE_RPC_DATA`. This information is intended for use by **pam_setcred()**.

During the call to **pam_acct_mgmt()**, the account modules may store data in the handle to indicate which passwords have aged. This information is intended for use by **pam_chauthtok()**.

The module can also store a cleanup function associated with the data. The PAM framework calls this cleanup function, when the application calls **pam_end()** to close the transaction.

Interaction with the User

The PAM service modules do not communicate directly with the user; instead they rely on the application to perform all such interactions. The application passes a pointer to the function, **conv()**, along with any associated application data pointers, through the `pam_conv` structure when it initiates an authentication transaction (via a call to **pam_start()**). The service module will then use the function, **conv()**, to prompt the user for data, output error messages, and display text information. Refer to **pam_start(3)** for more information. The modules are responsible for the localization of all messages to the user.

CONVENTIONS

By convention, applications that need to prompt for a user name should call **pam_set_item()** and set the value of `PAM_USER_PROMPT` before calling **pam_authenticate()**. The service module's **pam_sm_authenticate()** function

will then call `pam_get_user()` to prompt for the user name. Note that certain PAM service modules (such as a smart card module) may override the value of `PAM_USER_PROMPT` and pass in their own prompt.

Though the PAM framework enforces no rules about the module's names, location, options and such, there are certain conventions that all module providers are expected to follow.

By convention, the modules should be located in the `/usr/lib/security` directory. Additional modules may be located in `/opt/<pkg>/lib`.

By convention, the modules are named `pam_<service_name>_<module_type>.so.1`. If the given module implements more than one module type (for example, `pam_unix.so.1` module), then the `module_type` suffix should be dropped.

For every such module, there should be a corresponding manual page in section 5 which should describe the *module_type* it supports, the functionality of the module, along with the options it supports. The dependencies should be clearly identified to the system administrator. For example, it should be made clear whether this module is a stand-alone module or depends upon the presence of some other module. One should also specify whether this module should come before or after some other module in the stack.

By convention, the modules should support the following options:

`debug` Syslog debugging information at `LOG_DEBUG` level. Be careful as to not log any sensitive information such as passwords.

`nowarn` Turn off warning messages such as "password is about to expire."

In addition, it is recommended that the `auth` and the `password` module support the following options:

`use_first_pass` Instead of prompting the user for the password, use the user's initial password (entered when the user was authenticated to the first authentication module in the stack) for authentication. If the passwords do not match, or if no password has been entered, return failure and do not prompt the user for a password. Support for this scheme allows the user to type only one password for multiple schemes.

`try_first_pass` Instead of prompting the user for the password, use the user's initial password (entered when the user was authenticated to the first authentication

module in the stack) for authentication. If the passwords do not match, or if no password has been entered, prompt the user for a password after identifying which type of password (ie. UNIX, etc.) is being requested. Support for this scheme allows the user to try to use only one password for multiple schemes, and type multiple passwords only if necessary.

If an unsupported option is passed to the modules, it should syslog the error at LOG_ERR level.

The permission bits on the service module should be set such that it is not writable by either "group" or "other." The PAM framework will not load the module if the above permission rules are not followed.

ERROR LOGGING

If there are any errors, the modules should log them using `syslog(3)` at the LOG_ERR level.

RETURN VALUES

The PAM service module functions may return any of the PAM error numbers specified in the specific man pages. It can also return a PAM_IGNORE error number to mean that the PAM framework should ignore this module regardless of whether it is required, optional or sufficient. This error number is normally returned when the module does not want to deal with the given user at all.

ATTRIBUTES

See `attributes(5)` for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT Level | MT-Safe with exceptions |

SEE ALSO

`pam(3)`, `pam_authenticate(3)`, `pam_chauthtok(3)`, `pam_get_user(3)`, `pam_open_session(3)`, `pam_setcred(3)`, `pam_set_item(3)`, `pam_sm_authenticate(3)`, `pam_sm_chauthtok(3)`, `pam_sm_open_session(3)`, `pam_sm_setcred(3)`, `pam_start(3)`, `pam_strerror(3)`, `syslog(3)`, `pam.conf(4)`, `attributes(5)`

NOTES

The interfaces in `libpam()` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

| | | | | | | | |
|---|---|-------------------------------|--|---|--|-------------------------------|---------------------------|
| NAME | pam_sm_acct_mgmt – service provider implementation for pam_acct_mgmt | | | | | | |
| SYNOPSIS | <pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> #include <security/pam_modules.h></pre> <pre>int pam_sm_acct_mgmt(pam_handle_t *pamh, int flags, int argc, const char **argv);</pre> | | | | | | |
| DESCRIPTION | <p>In response to a call to <code>pam_acct_mgmt(3)</code>, the PAM framework calls <code>pam_sm_acct_mgmt()</code> from the modules listed in the <code>pam.conf(4)</code> file. The account management provider supplies the back-end functionality for this interface function. Applications should not call this API directly.</p> <p>The function <code>pam_sm_acct_mgmt()</code> determines whether or not the current user's account and password are valid. This includes checking for password and account expiration, and valid login times. The user in question is specified by a prior call to <code>pam_start()</code>, and is referenced by the authentication handle, <code>pamh</code>, which is passed as the first argument to <code>pam_sm_acct_mgmt()</code>. The following flags may be set in the <code>flags</code> field:</p> <table border="0"> <tr> <td style="padding-right: 20px;"><code>PAM_SILENT</code></td> <td>The account management service should not generate any messages.</td> </tr> <tr> <td style="padding-right: 20px;"><code>PAM_DISALLOW_NULL_AUTH Tok</code></td> <td>The account management service should return <code>PAM_NEW_AUTH Tok_REQD</code> if the user has a null authentication token.</td> </tr> </table> <p>The <code>argc</code> argument represents the number of module options passed in from the configuration file <code>pam.conf(4)</code>. <code>argv</code> specifies the module options, which are interpreted and processed by the account management service. Please refer to the specific module man pages for the various available <i>options</i>. If an unknown option is passed to the module, an error should be logged through <code>syslog(3)</code> and the option ignored.</p> <p>If an account management module determines that the user password has aged or expired, it should save this information as state in the authentication handle, <code>pamh</code>, using <code>pam_set_data()</code>. <code>pam_chauthok()</code> uses this information to determine which passwords have expired.</p> | <code>PAM_SILENT</code> | The account management service should not generate any messages. | <code>PAM_DISALLOW_NULL_AUTH Tok</code> | The account management service should return <code>PAM_NEW_AUTH Tok_REQD</code> if the user has a null authentication token. | | |
| <code>PAM_SILENT</code> | The account management service should not generate any messages. | | | | | | |
| <code>PAM_DISALLOW_NULL_AUTH Tok</code> | The account management service should return <code>PAM_NEW_AUTH Tok_REQD</code> if the user has a null authentication token. | | | | | | |
| RETURN VALUES | <p>If there are no restrictions to logging in, <code>PAM_SUCCESS</code> is returned. The following error values may also be returned upon error:</p> <table border="0"> <tr> <td style="padding-right: 20px;"><code>PAM_USER_UNKNOWN</code></td> <td>User not known to underlying authentication module.</td> </tr> <tr> <td style="padding-right: 20px;"><code>PAM_NEW_AUTH Tok_REQD</code></td> <td>New authentication token required.</td> </tr> <tr> <td style="padding-right: 20px;"><code>PAM_ACCT_EXPIRED</code></td> <td>User account has expired.</td> </tr> </table> | <code>PAM_USER_UNKNOWN</code> | User not known to underlying authentication module. | <code>PAM_NEW_AUTH Tok_REQD</code> | New authentication token required. | <code>PAM_ACCT_EXPIRED</code> | User account has expired. |
| <code>PAM_USER_UNKNOWN</code> | User not known to underlying authentication module. | | | | | | |
| <code>PAM_NEW_AUTH Tok_REQD</code> | New authentication token required. | | | | | | |
| <code>PAM_ACCT_EXPIRED</code> | User account has expired. | | | | | | |

PAM_PERM_DENIED User denied access to account at this time.

PAM_IGNORE Ignore underlying account module regardless of whether the control flag is *required*, *optional* or *sufficient*.

ATTRIBUTES

See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT Level | MT-Safe with exceptions |

SEE ALSO

pam(3), **pam_acct_mgmt(3)**, **pam_set_data(3)**, **pam_start(3)**, **syslog(3)**, **libpam(4)**, **pam.conf(4)**, **attributes(5)**

NOTES

The interfaces in **libpam()** are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

NAME pam_sm_authenticate – service provider implementation for pam_authenticate

SYNOPSIS

```
cc [ flag ...] file ... -lpam [ library ...]
#include <security/pam_appl.h>
#include <security/pam_modules.h>
```

DESCRIPTION

In response to a call to **pam_authenticate(3)**, the PAM framework calls **pam_sm_authenticate()** from the modules listed in the **pam.conf(4)** file. The authentication provider supplies the back-end functionality for this interface function.

The function, **pam_sm_authenticate()**, is called to verify the identity of the current user. The user is usually required to enter a password or similar authentication token depending upon the authentication scheme configured within the system. The user in question is specified by a prior call to **pam_start()**, and is referenced by the authentication handle, *pamh*.

If the user is unknown to the authentication service, the service module should mask this error and continue to prompt the user for a password. It should then return the error, **PAM_USER_UNKNOWN**.

The following flag may be passed in to **pam_sm_authenticate()**:

| | |
|-----------------------------------|--|
| PAM_SILENT | The authentication service should not generate any messages. |
| PAM_DISALLOW_NULL_AUTH Tok | The authentication service should return |
| PAM_AUTH_ERROR | The user has a null authentication token. |

The *argc* argument represents the number of module options passed in from the configuration file **pam.conf(4)**. *argv* specifies the module options, which are interpreted and processed by the authentication service. Please refer to the specific module man pages for the various available *options*. If any unknown option is passed in, the module should log the error and ignore the option.

Before returning, **pam_sm_authenticate()** should call **pam_get_item()** and retrieve **PAM_AUTHTOK**. If it has not been set before and the value is **NULL**, **pam_sm_authenticate()** should set it to the password entered by the user using **pam_set_item()**.

An authentication module may save the authentication status (success or reason for failure) as state in the authentication handle using **pam_set_data(3)**. This information is intended for use by **pam_setcred()**.

RETURN VALUES

Upon successful completion, `PAM_SUCCESS` must be returned. In addition, the following values may be returned:

| | |
|------------------------------------|---|
| <code>PAM_MAXTRIES</code> | Maximum number of authentication attempts exceeded. |
| <code>PAM_AUTH_ERR</code> | Authentication failure. |
| <code>PAM_CRED_INSUFFICIENT</code> | Cannot access authentication data due to insufficient credentials. |
| <code>PAM_AUTHINFO_UNAVAIL</code> | Underlying authentication service can not retrieve authentication information. |
| <code>PAM_USER_UNKNOWN</code> | User not known to underlying authentication module. |
| <code>PAM_IGNORE</code> | Ignore underlying authentication module regardless of whether the control flag is <i>required</i> , <i>optional</i> , or <i>sufficient</i> 1. |

ATTRIBUTES

See `attributes(5)` for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT Level | MT-Safe with exceptions |

SEE ALSO

`pam(3)`, `pam_authenticate(3)`, `pam_get_item(3)`, `pam_set_data(3)`, `pam_set_item(3)`, `pam_setcred(3)`, `pam_start(3)`, `libpam(4)`, `pam.conf(4)`, `attributes(5)`

NOTES

Modules should not retry the authentication in the event of a failure. Applications handle authentication retries and maintain the retry count. To limit the number of retries, the module can return a `PAM_MAXTRIES` error.

The interfaces in `libpam()` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

| | | | | | | | | | |
|-----------------------------|---|------------|--|-----------------------------|---|------------------|--|---------------------|---|
| NAME | pam_sm_chauthtok – service provider implementation for pam_chauthtok | | | | | | | | |
| SYNOPSIS | <pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> #include <security/pam_modules.h> int pam_sm_chauthtok(pam_handle_t *pamh, const int flags);</pre> | | | | | | | | |
| DESCRIPTION | <p>In response to a call to pam_chauthtok() the PAM framework calls pam_sm_chauthtok(3) from the modules listed in the pam.conf(4) file. The password management provider supplies the back-end functionality for this interface function.</p> <p>pam_sm_chauthtok() changes the authentication token associated with a particular user referenced by the authentication handle, <i>pamh</i>.</p> <p>The following flag may be passed in to pam_chauthtok():</p> <table border="0"> <tr> <td style="padding-right: 20px;">PAM_SILENT</td> <td>The password service should not generate any messages.</td> </tr> <tr> <td style="padding-right: 20px;">PAM_CHANGE_EXPIRED_AUTH Tok</td> <td>The password service should only update those passwords that have aged. If this flag is not passed, the password service should update all passwords.</td> </tr> <tr> <td style="padding-right: 20px;">PAM_PRELIM_CHECK</td> <td>The password service should only perform preliminary checks. No passwords should be updated.</td> </tr> <tr> <td style="padding-right: 20px;">PAM_UPDATE_AUTH Tok</td> <td>The password service should update passwords.</td> </tr> </table> <p>Note that PAM_PRELIM_CHECK and PAM_UPDATE_AUTH Tok cannot be set at the same time.</p> <p>Upon successful completion of the call, the authentication token of the user will be ready for change or will be changed, depending upon the flag, in accordance with the authentication scheme configured within the system.</p> <p>The <i>argc</i> argument represents the number of module options passed in from the configuration file pam.conf(4). <i>argv</i> specifies the module options, which are interpreted and processed by the password management service. Please refer to the specific module man pages for the various available <i>options</i>.</p> <p>It is the responsibility of pam_sm_chauthtok() to determine if the new password meets certain strength requirements. pam_sm_chauthtok() may continue to re-prompt the user (for a limited number of times) for a new password until the password entered meets the strength requirements.</p> | PAM_SILENT | The password service should not generate any messages. | PAM_CHANGE_EXPIRED_AUTH Tok | The password service should only update those passwords that have aged. If this flag is not passed, the password service should update all passwords. | PAM_PRELIM_CHECK | The password service should only perform preliminary checks. No passwords should be updated. | PAM_UPDATE_AUTH Tok | The password service should update passwords. |
| PAM_SILENT | The password service should not generate any messages. | | | | | | | | |
| PAM_CHANGE_EXPIRED_AUTH Tok | The password service should only update those passwords that have aged. If this flag is not passed, the password service should update all passwords. | | | | | | | | |
| PAM_PRELIM_CHECK | The password service should only perform preliminary checks. No passwords should be updated. | | | | | | | | |
| PAM_UPDATE_AUTH Tok | The password service should update passwords. | | | | | | | | |

Before returning, **pam_sm_chauthtok()** should call **pam_get_item()** and retrieve both `PAM_AUTHTOK` and `PAM_OLDAUTHOK`. If both are `NULL`, **pam_sm_chauthtok()** should set them to the new and old passwords as entered by the user.

RETURN VALUES

Upon successful completion, `PAM_SUCCESS` must be returned. The following values may also be returned:

| | |
|--|---|
| <code>PAM_PERM_DENIED</code> | No permission. |
| <code>PAM_AUTHTOK_ERR</code> | Authentication token manipulation error. |
| <code>PAM_AUTHTOK_RECOVERY_ERR</code> | Old authentication token cannot be recovered. |
| <code>PAM_AUTHTOK_LOCK_BUSY</code> | Authentication token lock busy. |
| <code>PAM_AUTHTOK_DISABLE_AGING</code> | Authentication token aging disabled. |
| <code>PAM_USER_UNKNOWN</code> | User unknown to password service. |
| <code>PAM_TRY_AGAIN</code> | Preliminary check by password service failed. |

ATTRIBUTES

See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT Level | MT-Safe with exceptions |

SEE ALSO

ping(1M), **pam(3)**, **pam_chauthtok(3)**, **pam_get_data(3)**, **pam_get_item(3)**, **pam_set_data(3)**, **libpam(4)**, **pam.conf(4)**, **attributes(5)**

NOTES

The PAM framework invokes the password services twice. The first time the modules are invoked with the flag, `PAM_PRELIM_CHECK`. During this stage, the password modules should only perform preliminary checks. For example, they may `ping` remote name services to see if they are ready for updates. If a password module detects a transient error such as a remote name service temporarily down, it should return `PAM_TRY_AGAIN` to the PAM framework, which will immediately return the error back to the application. If all password modules pass the preliminary check, the PAM framework invokes the password services again with the flag, `PAM_UPDATE_AUTHTOK`. During this

stage, each password module should proceed to update the appropriate password. Any error will again be reported back to application.

If a service module receives the flag `PAM_CHANGE_EXPIRED_AUTH Tok`, it should check whether the password has aged or expired. If the password has aged or expired, then the service module should proceed to update the password. If the status indicates that the password has not yet aged or expired, then the password module should return `PAM_IGNORE`.

If a user's password has aged or expired, a PAM account module could save this information as state in the authentication handle, *pamh*, using `pam_set_data()`. The related password management module could retrieve this information using `pam_get_data()` to determine whether or not it should prompt the user to update the password for this particular module.

The interfaces in `libpam()` are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

| | | | | | |
|------------------------------|--|------------------------------|---|-------------------------|---|
| NAME | pam_sm_open_session, pam_sm_close_session – service provider implementation for pam_open_session and pam_close_session | | | | |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lpam [<i>library</i> ...] #include <security/pam_appl.h> #include <security/pam_modules.h> int pam_sm_open_session(pam_handle_t * <i>pamh</i>, int <i>flags</i>, int <i>argc</i>, const char ** <i>argv</i>); int pam_sm_close_session(pam_handle_t * <i>pamh</i>, int <i>flags</i>, int <i>argc</i>, const char ** <i>argv</i>);</pre> | | | | |
| DESCRIPTION | <p>In response to a call to <code>pam_open_session(3)</code> and <code>pam_close_session(3)</code>, the PAM framework calls <code>pam_sm_open_session()</code> and <code>pam_sm_close_session()</code>, respectively from the modules listed in the <code>pam.conf(4)</code> file. The session management provider supplies the back-end functionality for this interface function.</p> <p><code>pam_sm_open_session()</code> is called to initiate session management. <code>pam_sm_close_session()</code> is invoked when a session has terminated. The argument <code>pamh</code> is an authentication handle. The following flag may be set in the <code>flags</code> field:</p> <p><code>PAM_SILENT</code> Session service should not generate any messages. The <code>argc</code> argument represents the number of module options passed in from the configuration file <code>pam.conf(4)</code>. <code>argv</code> specifies the module options, which are interpreted and processed by the session management service. If an unknown option is passed in, an error should be logged through <code>syslog(3)</code> and the option ignored.</p> | | | | |
| RETURN VALUES | <p>Upon successful completion, <code>PAM_SUCCESS</code> should be returned. The following values may also be returned upon error:</p> <table border="0"> <tr> <td style="padding-right: 20px;"><code>PAM_SESSION_ERR</code></td> <td>Cannot make or remove an entry for the specified session.</td> </tr> <tr> <td style="padding-right: 20px;"><code>PAM_IGNORE</code></td> <td>Ignore underlying session module regardless of whether the control flag is <i>required</i>, <i>optional</i> or <i>sufficient</i>.</td> </tr> </table> | <code>PAM_SESSION_ERR</code> | Cannot make or remove an entry for the specified session. | <code>PAM_IGNORE</code> | Ignore underlying session module regardless of whether the control flag is <i>required</i> , <i>optional</i> or <i>sufficient</i> . |
| <code>PAM_SESSION_ERR</code> | Cannot make or remove an entry for the specified session. | | | | |
| <code>PAM_IGNORE</code> | Ignore underlying session module regardless of whether the control flag is <i>required</i> , <i>optional</i> or <i>sufficient</i> . | | | | |

ATTRIBUTES

See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT Level | MT-Safe with exceptions |

SEE ALSO

pam(3) , **pam_open_session(3)** , **syslog(3)** , **libpam(4)** , **pam.conf(4)** , **attributes(5)**

NOTES

The interfaces in **libpam()** are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.

| | | | | | | | | | | | |
|-----------------------|--|--------------------|--|-----------------|---|-----------------------|--------------------------------|------------------|--------------------------------------|------------|---|
| NAME | pam_sm_setcred – service provider implementation for pam_setcred | | | | | | | | | | |
| SYNOPSIS | <pre>cc [flag ...] file ... -lpam [library ...] #include <security/pam_appl.h> #include <security/pam_modules.h></pre> <pre>int pam_sm_setcred(pam_handle_t *pamh, int flags, int argc, const char **argv);</pre> | | | | | | | | | | |
| DESCRIPTION | <p>In response to a call to pam_setcred(3), the PAM framework calls pam_sm_setcred() from the modules listed in the pam.conf(4) file. The authentication provider supplies the back-end functionality for this interface function.</p> <p>pam_sm_setcred() is called to set the credentials of the current user associated with the authentication handle, <i>pamh</i>. The following flags may be set in the <i>flags</i> field. Note that the first four flags are mutually exclusive:</p> <table border="0"> <tr> <td style="padding-right: 20px;">PAM_ESTABLISH_CRED</td> <td>Set user credentials for the authentication service.</td> </tr> <tr> <td>PAM_DELETE_CRED</td> <td>Delete user credentials associated with the authentication service.</td> </tr> <tr> <td>PAM_REINITIALIZE_CRED</td> <td>Reinitialize user credentials.</td> </tr> <tr> <td>PAM_REFRESH_CRED</td> <td>Extend lifetime of user credentials.</td> </tr> <tr> <td>PAM_SILENT</td> <td>Authentication service should not generate messages</td> </tr> </table> <p>If no flag is set, PAM_ESTABLISH_CRED is used as the default.</p> <p>The <i>argc</i> argument represents the number of module options passed in from the configuration file pam.conf(4). <i>argv</i> specifies the module options, which are interpreted and processed by the authentication service. If an unknown option is passed to the module, an error should be logged and the option ignored.</p> <p>If the PAM_SILENT flag is not set, then pam_sm_setcred() should print any failure status from the corresponding pam_sm_authenticate() function using the conversation function.</p> <p>The authentication status (success or reason for failure) is saved as module-specific state in the authentication handle by the authentication module. The status should be retrieved using pam_get_data(), and used to determine if user credentials should be set.</p> | PAM_ESTABLISH_CRED | Set user credentials for the authentication service. | PAM_DELETE_CRED | Delete user credentials associated with the authentication service. | PAM_REINITIALIZE_CRED | Reinitialize user credentials. | PAM_REFRESH_CRED | Extend lifetime of user credentials. | PAM_SILENT | Authentication service should not generate messages |
| PAM_ESTABLISH_CRED | Set user credentials for the authentication service. | | | | | | | | | | |
| PAM_DELETE_CRED | Delete user credentials associated with the authentication service. | | | | | | | | | | |
| PAM_REINITIALIZE_CRED | Reinitialize user credentials. | | | | | | | | | | |
| PAM_REFRESH_CRED | Extend lifetime of user credentials. | | | | | | | | | | |
| PAM_SILENT | Authentication service should not generate messages | | | | | | | | | | |
| RETURN VALUES | Upon successful completion, PAM_SUCCESS should be returned. The following values may also be returned upon error: | | | | | | | | | | |

| | |
|------------------|--|
| PAM_CRED_UNAVAIL | Underlying authentication service can not retrieve user credentials. |
| PAM_CRED_EXPIRED | User credentials have expired. |
| PAM_USER_UNKNOWN | User unknown to the authentication service. |
| PAM_CRED_ERR | Failure in setting user credentials. |
| PAM_IGNORE | Ignore underlying authentication module regardless of whether the control flag is <i>required</i> , <i>optional</i> , or <i>sufficient</i> . |

ATTRIBUTES

See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT Level | MT-Safe with exceptions |

SEE ALSO

pam(3), **pam_authenticate(3)**, **pam_get_data(3)**, **pam_setcred(3)**, **pam_sm_authenticate(3)**, **libpam(4)**, **pam.conf(4)**, **attributes(5)**

NOTES

pam_sm_setcred() is passed the same module options that are used by **pam_sm_authenticate()**.

The interfaces in **libpam()** are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.

| | |
|--------------------|---|
| NAME | pam_start, pam_end – authentication transaction routines for PAM |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lpam [<i>library</i> ...] #include <security/pam_appl.h> int pam_start(const char * <i>service</i>, const char * <i>user</i>, const struct pam_conv * <i>pam_conv</i>, pam_handle_t * *<i>pamh</i>); int pam_end(pam_handle_t * <i>pamh</i>, int <i>status</i>);</pre> |
| DESCRIPTION | <p>pam_start() is called to initiate an authentication transaction. pam_start() takes as arguments the name of the current service, <i>service</i> , the name of the user to be authenticated, <i>user</i> , the address of the conversation structure, <i>pam_conv</i> , and the address of a variable to be assigned the authentication handle <i>pamh</i> . Upon successful completion, <i>pamh</i> will refer to a PAM handle for use with subsequent calls to the authentication library.</p> <p>The structure <i>pam_conv</i> contains the address of the conversation function provided by the application. The underlying PAM service module invokes this function to output information to and retrieve input from the user. The <i>pam_conv</i> structure has the following entries:</p> <pre>struct pam_conv { int (*conv)();\011\011/* Conversation function */ void *appdata_ptr;\011/* Application data */ };</pre> <p>where</p> <pre>int conv(int num_msg, const struct pam_message **msg, struct pam_response **resp, void *appdata_ptr);</pre> |

The function `conv()` is called by a service module to hold a PAM conversation with the application or user. For window applications, the application can create a new pop-up window to be used by the interaction.

The parameter `num_msg` is the number of messages associated with the call. The parameter `msg` is a pointer to an array of length `num_msg` of the `pam_message` structure.

The structure `pam_message` is used to pass prompt, error message, or any text information from the authentication service to the application or user. It is the responsibility of the PAM service modules to localize the messages. The memory used by `pam_message` has to be allocated and freed by the PAM modules. The `pam_message` structure has the following entries:

```
struct pam_message{
    int      msg_style;
    char     *msg;
};
```

The message style, `msg_style`, can be set to one of the following values:

| | |
|----------------------------------|---|
| <code>PAM_PROMPT_ECHO_OFF</code> | Prompt user, disabling echoing of response. |
| <code>PAM_PROMPT_ECHO_ON</code> | Prompt user, enabling echoing of response. |
| <code>PAM_ERROR_MSG</code> | Print error message. |
| <code>PAM_TEXT_INFO</code> | Print general text information. |

The maximum size of the message and the response string is `PAM_MAX_MSG_SIZE` as defined in `<security/pam.appl.h>`.

The structure `pam_response` is used by the authentication service to get the user's response back from the application or user. The storage used by `pam_response` has to be allocated by the application and freed by the PAM modules. The `pam_response` structure has the following entries:

```
struct pam_response{
    \011char\011*resp;
    \011int\011resp_retcode;\011/* currently not used, */
    \011\011\011\011          /* should be set to 0 */
};
```

It is the responsibility of the conversation function to strip off NEWLINE characters for PAM_PROMPT_ECHO_OFF and PAM_PROMPT_ECHO_ON message styles, and to add NEWLINE characters (if appropriate) for PAM_ERROR_MSG and PAM_TEXT_INFO message styles.

appdata_ptr is an application data pointer which is passed by the application to the PAM service modules. Since the PAM modules pass it back through the conversation function, the applications can use this pointer to point to any application-specific data.

pam_end() is called to terminate the authentication transaction identified by *pamh* and to free any storage area allocated by the authentication module. The argument, *status*, is passed to the `cleanup()` function stored within the `pam` handle, and is used to determine what module-specific state must be purged. A cleanup function is attached to the handle by the underlying PAM modules through a call to **pam_set_item(3)** to free module specific data.

RETURN VALUES

Refer to **pam(3)** for information on error related return values.

ATTRIBUTES

See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT Level | MT-Safe with exceptions |

SEE ALSO

pam(3), **pam_acct_mgmt(3)**, **pam_authenticate(3)**, **pam_chauthtok(3)**, **pam_open_session(3)**, **pam_setcred(3)**, **pam_set_item(3)**, **pam_strerror(3)**, **attributes(5)**

NOTES

The interfaces in **libpam()** are MT-Safe only if each thread within the multithreaded application uses its own PAM handle.

NAME pam_strerror – get PAM error message string

SYNOPSIS

```
cc [ flag ... ] file ... -lpam [ library ... ]
#include <security/pam_appl.h>
```

```
const char *pam_strerror(pam_handle_t*pamh, int errnum);
```

DESCRIPTION **pam_strerror()** maps the PAM error number in *errnum* to a PAM error message string, and returns a pointer to that string. The application should not free or modify the string returned.

The *pamh* argument is the PAM handle obtained by a prior call to **pam_start()**. If **pam_start()** returns an error, a null PAM handle should be passed.

ERRORS **pam_strerror()** returns `NULL` if *errnum* is out-of-range.

ATTRIBUTES See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT Level | MT-Safe with exceptions |

SEE ALSO **pam(3)**, **pam_start(3)**, **attributes(5)**

NOTES The interfaces in **libpam()** are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

| NAME | panel_above, panel_below – panels deck traversal primitives | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lpanel -lcurses [<i>library</i> ..] #include <panel.h> PANEL * panel_above(PANEL * <i>panel</i>); PANEL * panel_below(PANEL * <i>panel</i>);</pre> | | | | |
| DESCRIPTION | <p>panel_above() returns a pointer to the panel just above <i>panel</i> , or NULL if <i>panel</i> is the top panel. panel_below() returns a pointer to the panel just below <i>panel</i> , or NULL if <i>panel</i> is the bottom panel.</p> <p>If NULL is passed for <i>panel</i> , panel_above() returns a pointer to the bottom panel in the deck, and panel_below() returns a pointer to the top panel in the deck.</p> | | | | |
| RETURN VALUES | NULL is returned if an error occurs. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | curses(3X) , panels(3X) , attributes(5) | | | | |
| NOTES | <p>These routines allow traversal of the deck of currently visible panels.</p> <p>The header <code><panel.h></code> automatically includes the header <code><curses.h></code> .</p> | | | | |

NAME panel_move, move_panel – move a panels window on the virtual screen

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lpanel

-lcurses
[
library
.. ]
#include <panel.h>

int move_panel(PANEL * panel, int starty, int startx);
```

DESCRIPTION **move_panel()** moves the `curses` window associated with *panel* so that its upper left-hand corner is at *starty*, *startx*. See usage note, below.

RETURN VALUES OK is returned if the routine completes successfully, otherwise ERR is returned.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO **curses(3X)**, **panel_update(3X)**, **panels(3X)**, **attributes(5)**

NOTES For `panels` windows, use **move_panel()** instead of the **mvwin()** `curses` routine. Otherwise, **update_panels()** will not properly update the virtual screen.

The header `<panel.h>` automatically includes the header `<curses.h>`.

NAME panel_new, new_panel, del_panel – create and destroy panels

SYNOPSIS

```
cc
[
  flag
  ... ]
file
...
-lpanel

-lcurses
[
  library
  .. ]
#include <panel.h>

PANEL * new_panel(WINDOW * win);

int del_panel(PANEL * panel);
```

DESCRIPTION **new_panel()** creates a new panel associated with *win* and returns the panel pointer. The new panel is placed on top of the panel deck.

del_panel() destroys *panel*, but not its associated window.

RETURN VALUES **new_panel()** returns NULL if an error occurs.

del_win() returns OK if successful, ERR otherwise.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO **curses(3X)**, **panel_update(3X)**, **panels(3X)**, **attributes(5)**

NOTES The header `<panel.h>` automatically includes the header `<curses.h>`.

NAME panels – character based panels package

SYNOPSIS #include <panel.h>

DESCRIPTION The panel library is built using the `curses` library, and any program using panels routines must call one of the `curses` initialization routines such as `initscr`. A program using these routines must be compiled with `-lpanel` and `-lcurses` on the `cc` command line.

The panels package gives the applications programmer a way to have depth relationships between `curses` windows; a `curses` window is associated with every panel. The panels routines allow `curses` windows to overlap without making visible the overlapped portions of underlying windows. The initial `curses` window, `stdscr`, lies beneath all panels. The set of currently visible panels is the *deck* of panels.

The panels package allows the applications programmer to create panels, fetch and set their associated windows, shuffle panels in the deck, and manipulate panels in other ways.

Routine Name Index

The following table lists each panels routine and the name of the manual page on which it is described.

| panels Routine Name | Manual Page Name |
|---------------------|-------------------|
| bottom_panel | panel_top(3X) |
| del_panel | panel_new(3X) |
| hide_panel | panel_show(3X) |
| move_panel | panel_move(3X) |
| new_panel | panel_new(3X) |
| panel_above | panel_above(3X) |
| panel_below | panel_above(3X) |
| panel_hidden | panel_show(3X) |
| panel_userptr | panel_userptr(3X) |
| panel_window | panel_window(3X) |
| replace_panel | panel_window(3X) |
| set_panel_userptr | panel_userptr(3X) |
| show_panel | panel_show(3X) |
| top_panel | panel_top(3X) |
| update_panels | panel_update(3X) |

RETURN VALUES

Each `panels` routine that returns a pointer to an object returns `NULL` if an error occurs. Each panel routine that returns an integer, returns `OK` if it executes successfully and `ERR` if it does not.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`curses(3X)`, `attributes(5)` and 3X pages whose names begin "panel_" for detailed routine descriptions.

NOTES

The header `<panel.h>` automatically includes the header `<curses.h>`.

| NAME | panel_show, show_panel, hide_panel, panel_hidden – panels deck manipulation routines | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lpanel -lcurses [<i>library</i> ..] #include <panel.h> int show_panel(PANEL * <i>panel</i>); int hide_panel(PANEL * <i>panel</i>); int panel_hidden(PANEL * <i>panel</i>);</pre> | | | | |
| DESCRIPTION | <p>show_panel() makes <i>panel</i>, previously hidden, visible and places it on top of the deck of panels.</p> <p>hide_panel() removes <i>panel</i> from the panel deck and, thus, hides it from view. The internal data structure of the panel is retained.</p> <p>panel_hidden() returns TRUE (1) or FALSE (0) indicating whether or not <i>panel</i> is in the deck of panels.</p> | | | | |
| RETURN VALUES | show_panel() and hide_panel() return the integer OK upon successful completion or ERR upon error. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | curses(3X) , panel_update(3X) , panels(3X) , attributes(5) | | | | |
| NOTES | The header <code><panel.h></code> automatically includes the header <code><curses.h></code> . | | | | |

| NAME | panel_top, top_panel, bottom_panel – panels deck manipulation routines | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lpanel -lcurses [<i>library</i> ..] #include <panel.h> int top_panel(PANEL * <i>panel</i>); int bottom_panel(PANEL * <i>panel</i>);</pre> | | | | |
| DESCRIPTION | <p>top_panel() pulls <i>panel</i> to the top of the desk of panels. It leaves the size, location, and contents of its associated window unchanged.</p> <p>bottom_panel() puts <i>panel</i> at the bottom of the deck of panels. It leaves the size, location, and contents of its associated window unchanged.</p> | | | | |
| RETURN VALUES | All of these routines return the integer OK upon successful completion or ERR upon error. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | curses(3X) , panel_update(3X) , panels(3X) , attributes(5) | | | | |
| NOTES | The header <panel.h> automatically includes the header <curses.h>. | | | | |

NAME panel_update, update_panels – panels virtual screen refresh routine

SYNOPSIS

```
cc
[
  flag
  ... ]
file
...
-lpanel

-lcurses
[
  library
  .. ]
#include <panel.h>

void update_panels(void);
```

DESCRIPTION **update_panels()** refreshes the virtual screen to reflect the depth relationships between the panels in the deck. The user must use the curses library call **doupdate()** (see **curs_refresh(3X)**) to refresh the physical screen.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO **curs_refresh(3X)** , **curses(3X)** , **panels(3X)** , **attributes(5)**

NOTES The header `<panel.h>` automatically includes the header `<curses.h>` .

NAME panel_userptr, set_panel_userptr – associate application data with a panels panel

SYNOPSIS

```
cc
[
flag
... ]
file
...
-lpanel

-lcurses
[
library
.. ]
#include <panel.h>

int set_panel_userptr(PANEL * panel, char * ptr);
char * panel_userptr(PANEL * panel);
```

DESCRIPTION Each panel has a user pointer available for maintaining relevant information. **set_panel_userptr()** sets the user pointer of *panel* to *ptr*. **panel_userptr()** returns the user pointer of *panel*.

RETURN VALUES set_panel_userptr returns OK if successful, ERR otherwise. panel_userptr returns NULL if there is no user pointer assigned to *panel*.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO **curses(3X)**, **panels(3X)**, **attributes(5)**

NOTES The header <panel.h> automatically includes the header <curses.h>.

NAME panel_window, replace_panel – get or set the current window of a panels panel

SYNOPSIS

```
cc
[
  flag
  ... ]
file
...
-lpanel

-lcurses
[
  library
  .. ]
#include <panel.h>

WINDOW * panel_window(PANEL * panel);

int replace_panel(PANEL * panel, WINDOW * win);
```

DESCRIPTION **panel_window()** returns a pointer to the window of *panel* .
replace_panel() replaces the current window of *panel* with *win* .

RETURN VALUES **panel_window()** returns NULL on failure.
replace_panel() returns OK on successful completion, ERR otherwise.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO **curses(3X)** , **panels(3X)** , **attributes(5)**

NOTES The header <panel.h> automatically includes the header <curses.h> .

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------|--|--------|---------|---|----------|---|----------|---|------------|---|-------------|---|---------------|---|-------------------|---|-----------|---|-------------|---|-----------------|---|------------------|---|------------|---|--------------|
| NAME | pathfind – search for named file in named directories | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lgen [<i>library ...</i>] #include <libgen.h> char *pathfind(const char *<i>path</i>, const char *<i>name</i>, const char *<i>mode</i>);</pre> | | | | | | | | | | | | | | | | | | | | | | | | | | |
| DESCRIPTION | <p>pathfind() searches the directories named in <i>path</i> for the file <i>name</i>. The directories named in <i>path</i> are separated by semicolons. <i>mode</i> is a string of option letters chosen from the set [<i>rwxfbcdpugks</i>]:</p> <table border="0"> <tr> <td style="padding-right: 20px;">Letter</td> <td>Meaning</td> </tr> <tr> <td>r</td> <td>readable</td> </tr> <tr> <td>w</td> <td>writable</td> </tr> <tr> <td>x</td> <td>executable</td> </tr> <tr> <td>f</td> <td>normal file</td> </tr> <tr> <td>b</td> <td>block special</td> </tr> <tr> <td>c</td> <td>character special</td> </tr> <tr> <td>d</td> <td>directory</td> </tr> <tr> <td>p</td> <td>FIFO (pipe)</td> </tr> <tr> <td>u</td> <td>set user ID bit</td> </tr> <tr> <td>g</td> <td>set group ID bit</td> </tr> <tr> <td>k</td> <td>sticky bit</td> </tr> <tr> <td>s</td> <td>size nonzero</td> </tr> </table> <p>Options read, write, and execute are checked relative to the real (not the effective) user ID and group ID of the current process.</p> <p>If the file <i>name</i>, with all the characteristics specified by <i>mode</i>, is found in any of the directories specified by <i>path</i>, then pathfind() returns a pointer to a string containing the member of <i>path</i>, followed by a slash character (/), followed by <i>name</i>.</p> <p>If <i>name</i> begins with a slash, it is treated as an absolute path name, and <i>path</i> is ignored.</p> <p>An empty <i>path</i> member is treated as the current directory. / is not prepended at the occurrence of the first match; rather, the unadorned <i>name</i> is returned.</p> | Letter | Meaning | r | readable | w | writable | x | executable | f | normal file | b | block special | c | character special | d | directory | p | FIFO (pipe) | u | set user ID bit | g | set group ID bit | k | sticky bit | s | size nonzero |
| Letter | Meaning | | | | | | | | | | | | | | | | | | | | | | | | | | |
| r | readable | | | | | | | | | | | | | | | | | | | | | | | | | | |
| w | writable | | | | | | | | | | | | | | | | | | | | | | | | | | |
| x | executable | | | | | | | | | | | | | | | | | | | | | | | | | | |
| f | normal file | | | | | | | | | | | | | | | | | | | | | | | | | | |
| b | block special | | | | | | | | | | | | | | | | | | | | | | | | | | |
| c | character special | | | | | | | | | | | | | | | | | | | | | | | | | | |
| d | directory | | | | | | | | | | | | | | | | | | | | | | | | | | |
| p | FIFO (pipe) | | | | | | | | | | | | | | | | | | | | | | | | | | |
| u | set user ID bit | | | | | | | | | | | | | | | | | | | | | | | | | | |
| g | set group ID bit | | | | | | | | | | | | | | | | | | | | | | | | | | |
| k | sticky bit | | | | | | | | | | | | | | | | | | | | | | | | | | |
| s | size nonzero | | | | | | | | | | | | | | | | | | | | | | | | | | |
| EXAMPLES | <p>EXAMPLE 1 Example of finding the <code>ls</code> command using the <code>PATH</code> environment variable.</p> <p>To find the <code>ls</code> command using the <code>PATH</code> environment variable:</p> <pre>pathfind (getenv ("PATH"), "ls", "rx")</pre> | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RETURN VALUES | If no match is found, <code>pathname</code> returns a null pointer, <code>((char *) 0)</code> . | | | | | | | | | | | | | | | | | | | | | | | | | | |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

sh(1), **test(1)**, **access(2)**, **mknod(2)**, **stat(2)**, **getenv(3C)**,
attributes(5)

NOTES

The string pointed to by the returned pointer is stored in an area that is reused on subsequent calls to **pathfind()**. The string should not be deallocated by the caller.

When compiling multi-thread applications, the **_REENTRANT** flag must be defined on the compile line. This flag should only be used in multi-thread applications.

| | | |
|----------------------|--|---|
| NAME | pechochar, pecho_wchar – add character and refresh window | |
| SYNOPSIS | <pre>#include <curses.h> int pechochar(WINDOW * pad, chtype ch); int pecho_wchar(WINDOW * pad, const chtype * wch);</pre> | |
| PARAMETERS | <i>pad</i> | Is a pointer to the pad in which the character is to be added. |
| | <i>ch</i> | Is a pointer to the character to be written to the pad. |
| | <i>wch</i> | Is a pointer to the complex character to be written to the pad. |
| DESCRIPTION | The pechochar() function is equivalent to calling waddch(3XC) followed by a call to prefresh(3XC) . The pecho_wchar() function is equivalent to calling wadd_wch(3XC) followed by a call to prefresh() . prefresh() reuses the last position of the pad on the screen for its parameters. | |
| RETURN VALUES | On success, these functions return OK . Otherwise, they return ERR . | |
| ERRORS | None. | |
| SEE ALSO | add_wch(3XC) , addch(3XC) , newpad(3XC) | |

NAME | perror, errno – print system error messages

SYNOPSIS | #include <stdio.h>

| void **perror**(const char * s);

| #include <errno.h>

| int errno;

DESCRIPTION | The **perror()** function produces a message on the standard error output (file descriptor 2) describing the last error encountered during a call to a system or library function. The argument string *s* is printed, followed by a colon and a blank, followed by the message and a NEWLINE character. If *s* is a null pointer or points to a null string, the colon is not printed. The argument string should include the name of the program that incurred the error. The error number is taken from the external variable `errno`, which is set when errors occur but not cleared when non-erroneous calls are made. See **intro(2)**.

USAGE | If the application is linked with `-lintl`, then messages printed from this function are in the native language specified by the LC_MESSAGES locale category. See **setlocale(3C)**.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **intro(2)**, **fmtmsg(3C)**, **gettext(3C)**, **setlocale(3C)**, **strerror(3C)**, **attributes(5)**

| | |
|--------------------|--|
| NAME | pfmt – display error message in standard format |
| SYNOPSIS | <pre>#include <pfmt.h> int pfmt(FILE *stream, long flags, char *format, ... /* arg */);</pre> |
| DESCRIPTION | <p>The pfmt() retrieves a format string from a locale-specific message database (unless <code>MM_NOGET</code> is specified) and uses it for <code>printf(3S)</code> style formatting of <i>args</i>. The output is displayed on <i>stream</i>.</p> <p>The pfmt() function encapsulates the output in the standard error message format (unless <code>MM_NOSTD</code> is specified, in which case the output is similar to printf()).</p> <p>If the printf() format string is to be retrieved from a message database, the <code>format</code> argument must have the following structure:</p> <pre><catalog>: <msgnum>: <defmsg>.</pre> <p>If <code>MM_NOGET</code> is specified, only the <i>defmsg</i> field must be specified.</p> <p>The <i>catalog</i> field is used to indicate the message database that contains the localized version of the format string. This field must be limited to 14 characters selected from the set of all characters values, excluding <code>\0</code> (null) and the ASCII codes for <code>/</code> (slash) and <code>:</code> (colon).</p> <p>The <i>msgnum</i> field is a positive number that indicates the index of the string into the message database.</p> <p>If the catalog does not exist in the locale (specified by the last call to setlocale(3C) using the <code>LC_ALL</code> or <code>LC_MESSAGES</code> categories), or if the message number is out of bound, pfmt() will attempt to retrieve the message from the C locale. If this second retrieval fails, pfmt() uses the <i>defmsg</i> field of the <code>format</code> argument.</p> <p>If <i>catalog</i> is omitted, pfmt() will attempt to retrieve the string from the default catalog specified by the last call to setcat(3C). In this case, the <code>format</code> argument has the following structure:</p> <pre>: <msgnum>: <defmsg>.</pre> <p>The pfmt() will output <code>Message not found!!\n</code> as format string if <i>catalog</i> is not a valid catalog name, if no catalog is specified (either explicitly or with setcat()), if <i>msgnum</i> is not a valid number, or if no message could be retrieved from the message databases and <i>defmsg</i> was omitted.</p> |

The *flags* argument determine the type of output (such as whether the *format* should be interpreted as is or encapsulated in the standard message format), and the access to message catalogs to retrieve a localized version of *format*.

The *flags* argument is composed of several groups, and can take the following values (one from each group):

Output format control

MM_NOSTD Do not use the standard message format, interpret *format* as **printf()** format. Only *catalog access control flags* should be specified if **MM_NOSTD** is used; all other flags will be ignored.

MM_STD Output using the standard message format (default value 0).

Catalog access control

MM_NOGET Do not retrieve a localized version of *format*. In this case, only the *defmsg* field of the *format* is specified.

MM_GET Retrieve a localized version of *format* from the *catalog*, using *msgid* as the index and *defmsg* as the default message (default value 0).

Severity (standard message format only)

MM_HALT Generate a localized version of **HALT**, but do not halt the machine.

MM_ERROR Generate a localized version of **ERROR** (default value 0).

MM_WARNING Generate a localized version of **WARNING**.

MM_INFO Generate a localized version of **INFO**.

Additional severities can be defined. Add-on severities can be defined with number-string pairs with numeric values from the range [5-255], using **addsev(3C)**. The specified severity will be generated from the bitwise OR operation of the numeric value and other *flags*. If the severity is not defined, **pfmt()** uses the string **SEV=N**, where *N* is replaced by the integer severity value passed in *flags*.

Multiple severities passed in *flags* will not be detected as an error. Any combination of severities will be summed and the numeric value will cause the display of either a severity string (if defined) or the string **SEV=N** (if undefined).

Action

MM_ACTION Specify an action message. Any severity value is superseded and replaced by a localized version of **TO FIX**.

**STANDARD
ERROR MESSAGE
FORMAT**

The **pfmt()** function displays error messages in the following format:

```
label: severity: text
```

If no *label* was defined by a call to **setlabel(3C)**, the message is displayed in the format:

```
severity: text
```

If **pfmt()** is called twice to display an error message and a helpful *action* or recovery message, the output can look like:

```
label: severity: textlabel: TO FIX: text
```

RETURN VALUES

Upon success, **pfmt()** returns the number of bytes transmitted. Upon failure, it returns a negative value:

-1 Write error to *stream*.

EXAMPLES

EXAMPLE 1 Example of **pfmt()** function.

Example 1:

```
setlabel("UX:test");
pfmt(stderr, MM_ERROR, "test:2:Cannot open file: %s\n", strerror(errno));
displays the message:
UX:test: ERROR: Cannot open file: No such file or directory
```

Example 2:

```
setlabel("UX:test");
setcat("test");
pfmt(stderr, MM_ERROR, ":10:Syntax error\n");
pfmt(stderr, MM_ACTION, "55:Usage ...\n");
```

displays the message

```
UX:test: ERROR: Syntax error
UX:test: TO FIX: Usage ...
```

USAGE Since it uses `gettxt(3C)`, `pfmt()` should not be used.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-safe |

SEE ALSO `addsev(3C)`, `gettxt(3C)`, `lfmt(3C)`, `printf(3S)`, `setcat(3C)`, `setlabel(3C)`, `setlocale(3C)`, `attributes(5)`, `environ(5)`

| | |
|----------------------|--|
| NAME | plock – lock or unlock into memory process, text, or data |
| SYNOPSIS | #include <sys/lock.h> int plock(int <i>op</i>); |
| DESCRIPTION | <p>The plock() function allows the calling process to lock or unlock into memory its text segment (text lock), its data segment (data lock), or both its text and data segments (process lock). Locked segments are immune to all routine swapping. The effective user ID of the calling process must be super-user to use this call.</p> <p>The plock() function performs the function specified by <i>op</i>:</p> <p>PROCLOCK Lock text and data segments into memory (process lock).</p> <p>TXTLOCK Lock text segment into memory (text lock).</p> <p>DATLOCK Lock data segment into memory (data lock).</p> <p>UNLOCK Remove locks.</p> |
| RETURN VALUES | Upon successful completion, 0 is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error. |
| ERRORS | <p>The plock() function fails and does not perform the requested operation if:</p> <p>EAGAIN Not enough memory.</p> <p>EINVAL The <i>op</i> argument is equal to PROCLOCK and a process lock, a text lock, or a data lock already exists on the calling process; the <i>op</i> argument is equal to TXTLOCK and a text lock or a process lock already exists on the calling process; the <i>op</i> argument is equal to DATLOCK and a data lock or a process lock already exists on the calling process; or the <i>op</i> argument is equal to UNLOCK and no lock exists on the calling process.</p> <p>EPERM The effective user of the calling process is not super-user.</p> |
| USAGE | The mlock(3C) and mlockall(3C) functions are the preferred interfaces for process locking. |
| SEE ALSO | exec(2) , exit(2) , fork(2) , memcntl(2) , mlock(3C) , mlockall(3C) |

| | |
|--------------------|--|
| NAME | plot, arc, box, circle, closepl, closevt, cont, erase, label, line, linmod, move, openpl, openvt, point, space – graphics interface |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lplot [<i>library</i> ...] #include <plot.h> void arc(short x0, short y0, short x1, short y1, short x2, short y2); void box(short x0, short y0, short x1, short y1); void circle(short x, short y, short r); void closepl(); void closevt(); void cont(short x, short y); void erase(); void label(char * s); void line(short x0, short y0, short x1, short y1); void linmod(char * s); void move(short x, short y); void openpl(); void openvt(); void point(short x, short y); void space(short x0, short y0, short x1, short y1);</pre> |
| DESCRIPTION | <p>These functions generate graphics output for a set of output devices. The format of the output is dependent upon which link editor option is used when the program is compiled and linked (see <code>Link Editor</code>).</p> <p>The term "current point" refers to the current setting for the x and y coordinates.</p> |

The **arc()** function specifies a circular arc. The coordinates (x_0 , y_0) specify the center of the arc. The coordinates (x_1 , y_1) specify the starting point of the arc. The coordinates (x_2 , y_2) specify the end point of the circular arc.

The **box()** function specifies a rectangle with coordinates (x_0 , y_0) , (x_0 , y_1) , (x_1 , y_0) , and (x_1 , y_1) . The current point is set to (x_1 , y_1) .

The **circle()** function specifies a circle with a center at the coordinates (x , y) and a radius of r .

The **closevt()** and **closepl()** functions flush the output.

The **cont()** function specifies a line beginning at the current point and ending at the coordinates (x , y) . The current point is set to (x , y) .

The **erase()** function starts another frame of output.

The **label()** function places the null terminated string s so that the first character falls on the current point. The string is then terminated by a NEWLINE character.

The **line()** function draws a line starting at the coordinates (x_0 , y_0) and ending at the coordinates (x_1 , y_1) . The current point is set to (x_1 , y_1) .

The **linmod()** function specifies the style for drawing future lines. s may contain one of the following: dotted , solid , longdashed , shortdashed , or dotdashed .

The **move()** function sets the current point to the coordinates (x , y) .

The **openpl()** or **openvt()** function must be called to open the device before any other `plot` functions are called.

The **point()** function plots the point given by the coordinates (x , y) . The current point is set to (x , y) .

The **space()** function specifies the size of the plotting area. The plot will be reduced or enlarged as necessary to fit the area specified. The coordinates (x_0 , y_0) specify the lower left hand corner of the plotting area. The coordinates (x_1 , y_1) specify the upper right hand corner of the plotting area.

Link Editor

Various flavors of these functions exist for different output devices. They are obtained by using the following `ld(1)` options:

```
-lplot          device-independent graphics stream on standard output in
                the format described in plot(4B)
-l300           GSI 300 terminal
```

FILES

```

-1300s      GSI 300S terminal
-14014     Tektronix 4014 terminal
-1450      GSI 450 terminal
-lvt0

/usr/lib/libplot.a      archive library
/usr/lib/libplot.so.1   shared object
/usr/lib/sparcv9/libplot.so.1  64-bit shared object
/usr/lib/lib300.a      archive library
/usr/lib/lib300.so.1   shared object
/usr/lib/sparcv9/lib300.so.1  64-bit shared object

/usr/lib/lib300s.a     archive library
/usr/lib/lib300s.so.1 shared object
/usr/lib/sparcv9/lib300s.so.1 64-bit shared object

/usr/lib/lib4014.a    archive library
/usr/lib/lib4014.so.1 shared object
/usr/lib/sparcv9/lib4014.so.1 64-bit shared object

/usr/lib/lib450.a     archive library
/usr/lib/lib450.so.1  shared object
/usr/lib/sparcv9/lib450.so.1 64-bit shared object

/usr/lib/libvt0.a     archive library
/usr/lib/libvt0.so.1  shared object

```

/usr/lib/sparcv9/libvt64.so shared object

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

graph(1), **ld(1)**, **libplot(4)**, **plot(4B)**, **attributes(5)**

| | |
|----------------------|--|
| NAME | popen, pclose – initiate a pipe to or from a process |
| SYNOPSIS | <pre>#include <stdio.h> FILE * popen(const char * command, const char * mode); int pclose(FILE * stream);</pre> |
| DESCRIPTION | <p>The popen() function creates a pipe between the calling program and the command to be executed. The arguments to popen() are pointers to null-terminated strings. The <i>command</i> argument consists of a shell command line. The <i>mode</i> argument is an I/O mode, either <i>r</i> for reading or <i>w</i> for writing. The value returned is a stream pointer such that one can write to the standard input of the command, if the I/O mode is <i>w</i>, by writing to the file <i>stream</i> (see intro(3)); and one can read from the standard output of the command, if the I/O mode is <i>r</i>, by reading from the file <i>stream</i>. Because open files are shared, a type <i>r</i> command may be used as an input filter and a type <i>w</i> as an output filter.</p> <p>The environment of the executed command will be as if a child process were created within the popen() call using fork(2). If the application is standard-conforming (see standards(5)), the child is invoked with the call:</p> <pre>execl("/usr/bin/ksh", "ksh", "-c", command, (char *)0);</pre> <p>otherwise, the child is invoked with the call:</p> <pre>execl("/usr/bin/sh", "sh", "-c", command, (char *)0);</pre> <p>A stream opened by popen() should be closed by pclose(), which closes the pipe, and waits for the associated process to terminate and returns the termination status of the process running the command language interpreter. This is the value returned by waitpid(2). See wstat(5) for more information on termination status.</p> |
| RETURN VALUES | <p>The popen() function returns a null pointer if files or processes cannot be created.</p> <p>The pclose() function returns the termination status of the command. It returns <code>-1</code> if <i>stream</i> is not associated with a popen() command and sets <code>errno</code> to indicate the error.</p> |

ERRORS

The **popen()** function may fail if:

EMFILE There are currently `FOPEN_MAX` or `STREAM_MAX` streams open in the calling process.

EINVAL The *mode* argument is invalid.

The **pclose()** function will fail if:

ECHILD The status of the child process could not be obtained, as described above.

The **popen()** function may also set `errno` values as described by `fork(2)` or `pipe(2)`.

USAGE

If the original and **popen()** processes concurrently read or write a common file, neither should use buffered I/O. Problems with an output filter may be forestalled by careful buffer flushing, for example, with **fflush()** (see `fclose(3S)`). A security hole exists through the `IFS` and `PATH` environment variables. Full pathnames should be used (or `PATH` reset) and `IFS` should be set to space and tab (" \t").

EXAMPLES**EXAMPLE 1 popen() example**

The following program will print on the standard output (see `stdio(3S)`) the names of files in the current directory with a `.c` suffix.

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    char *cmd = "/usr/bin/ls *.c";
    char buf[BUFSIZ];
    FILE *ptr;

    if ((ptr = popen(cmd, "r")) != NULL)
        while (fgets(buf, BUFSIZ, ptr) != NULL)
            (void) printf("%s", buf);

    return 0;
}
```

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`ksh(1)`, `pipe(2)`, `wait(2)`, `waitpid(2)`, `fclose(3S)`, `fopen(3S)`, `stdio(3S)`, `system(3S)`, `attributes(5)`, `wstat(5)`, `standards(5)`

NAME pow – power function

SYNOPSIS cc [*flag ...*] *file ...* -lm [*library ...*]
#include <math.h>

double pow(double x, double y);

DESCRIPTION The **pow()** function computes the value of x raised to the power y , x^y . If x is negative, y must be an integer value.

RETURN VALUES Upon successful completion, **pow()** returns the value of x raised to the power y .

If x is 0 and y is 0, 1.0 is returned.

If y is NaN, or y is non-zero and x is NaN, NaN is returned. If y is 0.0 and x is NaN, NaN is returned.

If x is 0.0 and y is negative, `-HUGE_VAL` is returned and `errno` may be set to `EDOM` or `ERANGE`.

If the correct value would cause overflow, `±HUGE_VAL` is returned, and `errno` is set to `ERANGE`.

If the correct value would cause underflow to 0, 0 is returned and `errno` may be set to `ERANGE`.

For exceptional cases, **matherr(3M)** tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS The **pow()** function will fail if:

EDOM The value of x is negative and y is non-integral.

ERANGE The value to be returned would have caused overflow.

The **pow()** function may fail if:

EDOM The value of x is 0.0 and y is negative.

ERANGE The correct value would cause underflow.

USAGE An application wishing to check for error situations should set `errno` to 0 before calling **pow()**. If `errno` is non-zero on return, or the return value is NaN, an error has occurred.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

pow(3M)

Mathematical Library

SEE ALSO `exp(3M)`, `isnan(3M)`, `matherr(3M)`, `attributes(5)`, `standards(5)`

| | |
|-----------------|---|
| NAME | printf, fprintf, sprintf, vprintf, fprintf, vsprintf – formatted output conversion |
| SYNOPSIS | <pre>/usr/ucb/cc [<i>flag</i> ...] <i>file</i> ... #include <stdio.h> int printf(<i>format</i>, ...); const char * <i>format</i> ; int fprintf(<i>stream</i>, <i>format</i>, <i>va_list</i>); FILE * <i>stream</i> ; char * <i>format</i> ; <i>va_dcl</i> ; char * sprintf(<i>s</i>, <i>format</i>, <i>va_list</i>); char * <i>s</i> , * <i>format</i> ; <i>va_dcl</i> ; int vprintf(<i>format</i>, <i>ap</i>); char * <i>format</i> ; ; <i>va_list</i> <i>ap</i> ; int fprintf(<i>stream</i>, <i>format</i>, <i>ap</i>);</pre> |

```

FILE *
stream
;
char *
format
;
va_list
ap
;

char * vsprintf( s, format, ap);

char *
s
, *
format
;
va_list
ap
;

```

DESCRIPTION

printf() places output on the standard output stream `stdout`. **fprintf()** places output on the named output `stream`. **sprintf()** places “output,” followed by the NULL character (`\\0`), in consecutive bytes starting at `*s`; it is the user’s responsibility to ensure that enough storage is available.

vprintf(), **vfprintf()**, and **vsprintf()** are the same as **printf()**, **fprintf()**, and **sprintf()** respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by `varargs(5)`.

Each of these functions converts, formats, and prints its `arg`s under control of the `format`. The `format` is a character string which contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which causes conversion and printing of zero or more `arg`s. The results are undefined if there are insufficient `arg`s for the format. If the format is exhausted while `arg`s remain, the excess `arg`s are simply ignored.

Each conversion specification is introduced by the character `%`. After the `%`, the following appear in sequence:

- Zero or more `flags`, which modify the meaning of the conversion specification.
- An optional decimal digit string specifying a minimum `field width`. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left-adjustment flag ‘`-`’, described below, has

been given) to the field width. The padding is with blanks unless the field width digit string starts with a zero, in which case the padding is with zeros.

- A *precision* that gives the minimum number of digits to appear for the *d*, *i*, *o*, *u*, *x*, or *X* conversions, the number of digits to appear after the decimal point for the *e*, *E*, and *f* conversions, the maximum number of significant digits for the *g* and *G* conversion, or the maximum number of characters to be printed from a string in *s* conversion. The precision takes the form of a period (.) followed by a decimal digit string; a NULL digit string is treated as zero. Padding specified by the precision overrides the padding specified by the field width.
- An optional *l* (ell) specifying that a following *d*, *i*, *o*, *u*, *x*, or *X* conversion character applies to a long integer *arg*. An *l* before any other conversion character is ignored.
- A character that indicates the type of conversion to be applied.

A field width or precision or both may be indicated by an asterisk (*) instead of a digit string. In this case, an integer *arg* supplies the field width or precision. The *arg* that is actually converted is not fetched until the conversion letter is seen, so the *arg* s specifying field width or precision must appear *before* the *arg* (if any) to be converted. A negative field width argument is taken as a ' - ' flag followed by a positive field width. If the precision argument is negative, it will be changed to zero.

The flag characters and their meanings are:

- The result of the conversion will be left-justified within the field.
- + The result of a signed conversion will always begin with a sign (+ or -).
- blank** If the first character of a signed conversion is not a sign, a blank will be prefixed to the result. This implies that if the blank and + flags both appear, the blank flag will be ignored.
- # This flag specifies that the value is to be converted to an "alternate form." For *c*, *d*, *i*, *s*, and *u* conversions, the flag has no effect. For *o* conversion, it increases the precision to force the first digit of the result to be a zero. For *x* or *X* conversion, a non-zero result will have 0x or 0X prefixed to it. For *e*, *E*, *f*, *g*, and *G* conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point appears in the result of these

conversions only if a digit follows it). For `g` and `G` conversions, trailing zeroes will *not* be removed from the result (which they normally are).

The conversion characters and their meanings are:

`d, i, o, u, x, X` The integer *arg* is converted to signed decimal (`d` or `i`), unsigned octal (`o`), unsigned decimal (`u`), or unsigned hexadecimal notation (`x` and `X`), respectively; the letters `abcdef` are used for `x` conversion and the letters `ABCDEF` for `X` conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeroes. (For compatibility with older versions, padding with leading zeroes may alternatively be specified by prepending a zero to the field width. This does not imply an octal value for the field width.) The default precision is 1. The result of converting a zero value with a precision of zero is a `NULL` string.

`f` The float or double *arg* is converted to decimal notation in the style `[-]ddd . ddd` where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed.

`e, E` The float or double *arg* is converted in the style `[-]d . ddd e± ddd`, where there is one digit before the decimal point and the number of digits after it is equal to the precision; when the precision is missing, 6 digits are produced; if the precision is zero, no decimal point appears. The `E` format code will produce a number with `E` instead of `e` introducing the exponent. The exponent always contains at least two digits.

`g, G` The float or double *arg* is printed in style `f` or `e` (or in style `E` in the case of a `G` format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style `e` or `E` will be used only if the exponent resulting from the conversion is less than `-4` or greater than the precision. Trailing zeroes are removed from the result; a decimal point appears only if it is followed by a digit.

The `e, E, f, g,` and `G` formats print IEEE indeterminate values (infinity or not-a-number) as "Infinity" or "NaN" respectively.

- c** The character *arg* is printed.
- s** The *arg* is taken to be a string (character pointer) and characters from the string are printed until a NULL character (`\\0`) is encountered or until the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first NULL character are printed. A NULL value for *arg* will yield undefined results.
- %** Print a % ; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Padding takes place only if the specified field width exceeds the actual width. Characters generated by **printf()** and **fprintf()** are printed as if **putc(3S)** had been called.

RETURN VALUES

Upon success, **printf()** and **fprintf()** return the number of characters transmitted, excluding the null character. **vprintf()** and **vfprintf()** return the number of characters transmitted. **sprintf()** and **vsprintf()** always return *s* . If an output error is encountered, **printf()** , **fprintf()** , **vprintf()** , and **vfprintf()** return EOF.

EXAMPLES

EXAMPLE 1 Examples of the **printf** Command To Print a Date and Time

To print a date and time in the form "Sunday, July 3, 10:02," where *weekday* and *month* are pointers to NULL-terminated strings:

```
printf("%s, %s %i, %d:%.2d", weekday, month, day, hour, min);
```

CODE EXAMPLE 1 Examples of the **printf** Command To Print to Five Decimal Places

To print to five decimal places:

```
printf("pi = %.5f", 4 * atan(1. 0));
```

SEE ALSO

econvert(3) , **putc(3S)** , **scanf(3S)** , **vprintf(3S)** , **varargs(5)**

NOTES

Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

printf(3B)

SunOS/BSD Compatibility Library Functions

Very wide fields (>128 characters) fail.

| | |
|--------------------|--|
| NAME | printf, fprintf, sprintf, snprintf – print formatted output |
| SYNOPSIS | <pre>#include <stdio.h> int printf(const char * format, /* args */ ...); int fprintf(FILE * stream, const char * format, /* args */ ...); int sprintf(char * s, const char * format, /* args */ ...); int snprintf(char * s, size_t n, const char * format, /* args */ ...);</pre> |
| DESCRIPTION | <p>The printf() function places output on the standard output stream <code>stdout</code>.</p> <p>The fprintf() function places output on on the named output stream <code>stream</code>.</p> <p>The sprintf() function places output, followed by the null byte (<code>\\0</code>), in consecutive bytes starting at <code>s</code>; it is the user's responsibility to ensure that enough storage is available.</p> <p>The snprintf() function is identical to sprintf() with the addition of the argument <code>n</code>, which specifies the size of the buffer referred to by <code>s</code>.</p> <p>Each of these functions converts, formats, and prints its <i>args</i> under control of the <i>format</i>. The <i>format</i> is a character string, beginning and ending in its initial shift state, if any. The <i>format</i> is composed of zero or more directives defined as follows:</p> <ul style="list-style-type: none"> ■ <i>ordinary characters</i>, which are simply copied to the output stream; ■ <i>escape sequences</i>, which represent non-graphic characters; and ■ <i>conversion specifications</i>, each of which results in the fetching of zero or more arguments. <p>The results are undefined if there are insufficient arguments for the <i>format</i>. If the <i>format</i> is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.</p> <p>Conversions can be applied to the <i>n</i>th argument after the <i>format</i> in the argument list, rather than to the next unused argument. In this case, the conversion character <code>%</code> (see below) is replaced by the sequence <code>% n \$</code>, where <i>n</i> is a decimal integer in the range <code>[1, NL_ARGMAX]</code>, giving the position of the argument in the argument list. This feature provides for the definition of format strings that select arguments in an order appropriate to specific languages (see the <code>EXAMPLES</code> section).</p> <p>In format strings containing the <code>% n \$</code> form of conversion specifications, numbered arguments in the argument list can be referenced from the format string as many times as required.</p> |

| | |
|---------------------------|---|
| Escape Sequences | <p>In format strings containing the % form of conversion specifications, each argument in the argument list is used exactly once.</p> <p>All forms of the printf() functions allow for the insertion of a language-dependent radix character. The radix character is defined by the program's locale (category LC_NUMERIC). In the POSIX locale, or in a locale where the radix character is not defined, the radix character defaults to a period (.).</p> <p>The following escape sequences produce the associated action on display devices capable of the action:</p> <p>\\a Alert. Ring the bell.</p> <p>\\b Backspace. Move the printing position to one character before the current position, unless the current position is the start of a line.</p> <p>\\f Form feed. Move the printing position to the initial printing position of the next logical page.</p> <p>\\ Newline. Move the printing position to the start of the next line.</p> <p>\\r Carriage return. Move the printing position to the start of the current line.</p> <p>\\t Horizontal tab. Move the printing position to the next implementation-defined horizontal tab position on the current line.</p> <p>\\v Vertical tab. Move the printing position to the start of the next implementation-defined vertical tab position.</p> |
| Conversion Specifications | <p>Each conversion specification is introduced by the character % or by the character sequence % n \$, after which the following appear in sequence:</p> <ul style="list-style-type: none"> ■ An optional field, consisting of a decimal digit string followed by a \$, specifying the next <i>args</i> to be converted. If this field is not provided, the <i>args</i> following the last <i>args</i> converted will be used. ■ Zero or more <i>flags</i> (in any order), which modify the meaning of the conversion specification. ■ An optional minimum <i>field width</i> . If the converted value has fewer bytes than the field width, it will be padded with spaces by default on the left; it will be padded on the right, if the left-adjustment flag (--), described below, is given to the field width. The field width takes the form of an asterisk (*), described below, or a decimal integer. |

If the conversion character is `s`, a standard-conforming application (see **standards(5)**) interprets the field width as the minimum number of bytes to be printed; an application that is not standard-conforming interprets the field width as the minimum number of columns of screen display. For an application that is not standard-conforming, `%10s` means if the converted value has a screen width of 7 columns, 3 spaces would be padded on the right.

If the format is `%ws`, then the field width should be interpreted as the minimum number of columns of screen display.

- An optional *precision* that gives the minimum number of digits to appear for the `d`, `i`, `o`, `u`, `x`, or `X` conversions (the field is padded with leading zeros); the number of digits to appear after the radix character for the `e`, `E`, and `f` conversions, the maximum number of significant digits for the `g` and `G` conversions; or the maximum number of bytes to be printed from a string in `s` and `S` conversions. The precision takes the form of a period (.) followed either by an asterisk (*), described below, or an optional decimal digit string, where a null digit string is treated as 0. If a precision appears with any other conversion character, the behavior is undefined.

If the conversion character is `s` or `S`, a standard-conforming application (see **standards(5)**) interprets the precision as the maximum number of bytes to be written; an application that is not standard-conforming interprets the precision as the maximum number of columns of screen display. For an application that is not standard-conforming, `%.5s` would print only the portion of the string that would display in 5 screen columns. Only complete characters are written.

For `%ws`, the precision should be interpreted as the maximum number of columns of screen display. The precision takes the form of a period (.) followed by a decimal digit string; a null digit string is treated as zero. Padding specified by the precision overrides the padding specified by the field width.

- An optional `h` specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion character applies to a type `short int` or `unsigned short int` argument (the argument will be promoted according to the integral promotions and its value converted to `short int` or `unsigned short int` before printing); an optional `h` specifies that a following `n` conversion character applies to a pointer to a type `short int` argument. An optional `l` (ell) specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion character applies to a type `long int` or `unsigned long int` argument; an optional `l` (ell) specifies that a following `n` conversion character applies to a pointer to a type `long int` argument. An optional `ll` (ell ell) specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion character applies to a type `long long` or `unsigned long long` argument; an optional `ll` (ell ell) specifies that a

following `n` conversion character applies to a pointer to a `long long` argument. An optional `L` specifies that a following `e`, `E`, `f`, `g`, or `G` conversion character applies to a `long double` argument. If an `h`, `l`, or `L` appears before any other conversion character, the behavior is undefined.

- An optional `l` (ell) specifying that a following `c` conversion character applies to a `wint_t` argument; an optional `l` (ell) specifying that a following `s` conversion character applies to a pointer to a `wchar_t` argument.
- A *conversion character* (see below) that indicates the type of conversion to be applied.

A field width or precision may be indicated by an asterisk (`*`) instead of a digit string. In this case, an integer *args* supplies the field width or precision. The *args* that is actually converted is not fetched until the conversion letter is seen, so the *args* specifying field width or precision must appear before the *args* (if any) to be converted. If the *precision* argument is negative, it will be changed to zero. A negative field width argument is taken as a `-` flag, followed by a positive field width. A negative precision is taken as if the precision were omitted. In format strings containing the `% n $` form of a conversion specification, a field width or precision may be indicated by the sequence `* m $`, where *m* is a decimal integer in the range `[1, NL_ARGMAX]` giving the position in the argument list (after the format argument) of an integer argument containing the field width or precision, for example:

```
printf("%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```

The *format* can contain either numbered argument specifications (that is, `% n $` and `* m $`), or unnumbered argument specifications (that is, `%` and `*`), but normally not both. The only exception to this is that `%%` can be mixed with the `% n $` form. The results of mixing numbered and unnumbered argument specifications in a *format* string are undefined. When numbered argument specifications are used, specifying the *N*th argument requires that all the leading arguments, from the first to the $(N-1)$ th, are specified in the format string.

Flag Characters

The flag characters and their meanings are:

- ' The integer portion of the result of a decimal conversion (`%i`, `%d`, `%u`, `%f`, `%g`, or `%G`) will be formatted with thousands' grouping characters. For other conversions the behavior is undefined. The non-monetary grouping character is used.
- The result of the conversion will be left-justified within the field. (It will be right-justified if this flag is not specified.)

| | |
|------------------------------|---|
| + | The result of a signed conversion will always begin with a sign (+ or -). (It will begin with a sign only when a negative value is converted if this flag is not specified.) |
| space | If the first character of a signed conversion is not a sign, a space will be placed before the result. This means that if the <code>space</code> and <code>+</code> flags both appear, the space flag will be ignored. |
| # | The value is to be converted to an alternate form. For <code>c</code> , <code>d</code> , <code>i</code> , <code>s</code> , and <code>u</code> conversions, the flag has no effect. For an <code>o</code> conversion, it increases the precision to force the first digit of the result to be a zero. For <code>x</code> (or <code>X</code>) conversion, a non-zero result will have <code>0x</code> (or <code>0X</code>) prepended to it. For <code>e</code> , <code>E</code> , <code>f</code> , <code>g</code> , and <code>G</code> conversions, the result will always contain a radix character, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For <code>g</code> and <code>G</code> conversions, trailing zeros will not be removed from the result as they normally are. |
| 0 | For <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , <code>X</code> , <code>e</code> , <code>E</code> , <code>f</code> , <code>g</code> , and <code>G</code> conversions, leading zeros (following any indication of sign or base) are used to pad to the field width; no space padding is performed. If the <code>0</code> and <code>-</code> flags both appear, the <code>0</code> flag will be ignored. For <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , and <code>X</code> conversions, if a precision is specified, the <code>0</code> flag will be ignored. For other conversions, the behavior is undefined. |
| Conversion Characters | Each conversion character results in fetching zero or more <i>args</i> . The results are undefined if there are insufficient <i>args</i> for the format. If the format is exhausted while <i>args</i> remain, the excess <i>args</i> are ignored. The conversion characters and their meanings are: |
| <code>d, i</code> | The <code>int</code> argument is converted to a signed decimal in the style <code>[-] dddd</code> . The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting 0 with an explicit precision of 0 is no characters. |
| <code>o</code> | The <code>unsigned int</code> argument is converted to unsigned octal format in the style <code>dddd</code> . The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting 0 with an explicit precision of 0 is no characters. |

- u** The `unsigned int` argument is converted to unsigned decimal format in the style `ddd`. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting 0 with an explicit precision of 0 is no characters.
- x** The `unsigned int` argument is converted to unsigned hexadecimal format in the style `ddd`; the letters `abcdef` are used. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting 0 with an explicit precision of 0 is no characters.
- X** Behaves the same as the `x` conversion character except that letters `ABCDEF` are used instead of `abcdef`.
- f** The double `args` is converted to decimal notation in the style `[-]ddd`. `ddd`, where the number of digits after the radix character (see `setlocale(3C)`) is equal to the precision specification. If the precision is omitted from `arg`, six digits are output; if the precision is explicitly zero and the `#` flag is not specified, no radix character appears. If a radix character appears, at least 1 digit appears before it. The value is rounded to the appropriate number of digits.
- e, E** The double `args` is converted to the style `[-]d . ddd e ±dd`, where there is one digit before the radix character (which is non-zero if the argument is non-zero) and the number of digits after it is equal to the precision. When the precision is missing, six digits are produced; if the precision is zero and the `#` flag is not specified, no radix character appears. The `E` conversion character will produce a number with `E` instead of `e` introducing the exponent. The exponent always contains at least two digits. The value is rounded to the appropriate number of digits.
- g, G** The double `args` is printed in style `f` or `e` (or in style `E` in the case of a `G` conversion character), with the precision specifying the number of significant digits. If the precision is zero, it is taken as one. The style used depends on the value converted: style `e` (or `E`) will be used only if the exponent resulting from the conversion is less than `-4` or greater than or equal to the precision. Trailing zeros are removed from the fractional part of the result. A radix character appears only if it is followed by a digit.

- c** The *int args* is converted to an `unsigned char`, and the resulting byte is printed.
- If an `l` (ell) qualifier is present, the `wint_t` argument is converted as if by an `ls` conversion specification with no precision and an argument that points to a two-element array of type `wchar_t`, the first element of which contains the `wint_t` argument to the `ls` conversion specification and the second element contains a null wide-character.
- C** Same as `lc`.
- wc** The *int args* is converted to a wide character (`wchar_t`), and the resulting wide character is printed.
- s** The *args* is taken to be a string (character pointer) and characters from the string are written up to (but not including) a terminating null byte. If a precision is specified, a standard-conforming application (see `standards(5)`) will write only the number of bytes specified by precision; an application that is not standard-conforming will write only the portion of the string that will display in the number of columns of screen display specified by precision.
- If the precision is not specified, it is taken to be infinite, so all bytes up to the first null byte are printed. A null value for *args* will yield undefined results.
- If an `l` (ell) qualifier is present, the argument must be a pointer to an array of type `wchar_t`. Wide-characters from the array are converted to characters (each as if by a call to the `wcrtomb(3C)` function, with the conversion state described by an `mbstate_t` object initialized to zero before the first wide-character is converted) up to and including a terminating null wide-character. The resulting characters are written up to (but not including) the terminating null character (byte). If no precision is specified, the array must contain a null wide-character. If a precision is specified, no more than that many characters (bytes) are written (including shift sequences, if any), and the array must contain a null wide-character if, to equal the character sequence length given by the precision, the function would need to access a wide-character one past the end of the array. In no case is a partial character written.
- S** Same as `ls`.
- ws** The *args* is taken to be a wide character string (wide character pointer) and wide characters from the string are written up to (but not including) a terminating null character; if the precision is specified,

only the portion of the wide character string that will display in the number of columns of screen display specified by precision will be written. If the precision is not specified, it is taken to be infinite, so all wide characters up to the first null character are printed. A null value for *args* will yield undefined results.

P The *args* should be a pointer to `void`. The value of the pointer is converted to a set of sequences of printable characters, which should be the same as the set of sequences that are matched by the `%p` conversion of the `scanf(3S)` function.

n The argument should be a pointer to an integer into which is written the number of bytes written to the output standard I/O stream so far by this call to `printf()`, `fprintf()`, or `sprintf()`. No argument is converted.

% Print a `%`; no argument is converted. The entire conversion specification must be `%%`.

If a conversion specification does not match one of the above forms, the behavior is undefined.

If a floating-point value is the internal representation for infinity, the output is `[±]Infinity`, where *Infinity* is either `Infinity` or `Inf`, depending on the desired output string length. Printing of the sign follows the rules described above.

If a floating-point value is the internal representation for “not-a-number,” the output is `[±]NaN`. Printing of the sign follows the rules described above.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by `printf()` and `fprintf()` are printed as if the `putc(3S)` function had been called.

The `st_ctime` and `st_mtime` fields of the file will be marked for update between the call to a successful execution of `printf()` or `fprintf()` and the next successful completion of a call to `fflush(3S)` or `fclose(3S)` on the same stream or a call to `exit(3C)` or `abort(3C)`.

RETURN VALUES

The `printf()`, `fprintf()`, and `sprintf()` functions return the number of bytes transmitted (not including the `\\0` in the case of `sprintf()`). The `snprintf()` function returns the number of characters formatted, that is, the number of characters that would have been written to the buffer if it were large enough. Each function returns a negative value if an output error was encountered.

ERRORS

For the conditions under which `printf()` and `fprintf()` will fail and may fail, refer to `fputc(3S)` or `fputwc(3S)`.

In addition, all forms of **printf()** may fail if:

EILSEQ A wide-character code that does not correspond to a valid character has been detected.

EINVAL There are insufficient arguments.

In addition, **printf()** and **fprintf()** may fail if:

ENOMEM Insufficient storage space is available.

USAGE

If the application calling the **printf()** functions has any objects of type `wint_t` or `wchar_t`, it must also include the header `<wchar.h>` to have these objects defined.

The **sprintf()** and **snprintf()** functions are MT-Safe in multithreaded applications. The **printf()** and **fprintf()** functions can be used safely in multithreaded applications, as long as **setlocale(3C)** is not being called to change the locale.

EXAMPLES

EXAMPLE 1 To print the language-independent date and time format, the following statement could be used:

```
printf (format, weekday, month, day, hour, min);
```

For American usage, *format* could be a pointer to the string:

```
"%s, %s %d, %d:%.2d\  
"
```

producing the message:

```
Sunday, July 3, 10:02
```

whereas for German usage, *format* could be a pointer to the string:

```
"%1$s, %3$d. %2$s, %4$d:%5$.2d\  
"
```

producing the message:

```
Sonntag, 3. Juli, 10:02
```

EXAMPLE 2 To print a date and time in the form Sunday, July 3, 10:02, where weekday and month are pointers to null-terminated strings:

```
printf("%s, %s %i, %d:%.2d", weekday, month, day, hour, min);
```

EXAMPLE 3 To print pi to 5 decimal places:

```
printf("pi = %.5f", 4 * atan(1.0));
```

Default

EXAMPLE 4 The following example applies only to applications which are not standard-conforming (see **standards(5)**). To print a list of names in columns which are 20 characters wide:

```
printf("%20s%20s%20s", lastname, firstname, middlename);
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |
| CSI | Enabled |

SEE ALSO

exit(2), **lseek(2)**, **write(2)**, **abort(3C)**, **ecvt(3C)**, **exit(3C)**, **fclose(3S)**, **fflush(3S)**, **fputc(3S)**, **fputwc(3S)**, **putc(3S)**, **scanf(3S)**, **setlocale(3C)**, **stdio(3S)**, **wcstombs(3C)**, **wctomb(3C)**, **attributes(5)**, **environ(5)**, **standards(5)**

| | |
|-----------------|---|
| NAME | proc_service – process service interfaces |
| SYNOPSIS | <pre> #include <proc_service.h> ps_err_e ps_pmodel(struct ps_prochandle *ph, int *data_model); ps_err_e ps_pglobal_lookup(struct ps_prochandle *ph, const char *object_name, const char *sym_name , psaddr_t *sym_addr); ps_err_e ps_pglobal_sym(struct ps_prochandle *ph, const char *object_name, const char *sym_name , ps_sym_t *sym); ps_err_e ps_pread(struct ps_prochandle *ph, psaddr_t addr, void *buf, size_t size); ps_err_e ps_pwrite(struct ps_prochandle *ph, psaddr_t addr, const void *buf, size_t size); ps_err_e ps_pdrread(struct ps_prochandle *ph, psaddr_t addr, void *buf, size_t size); ps_err_e ps_pdrwrite(struct ps_prochandle *ph, psaddr_t addr, const void *buf, size_t size); ps_err_e ps_ptread(struct ps_prochandle *ph, psaddr_t addr, void *buf, size_t size); ps_err_e ps_ptwrite(struct ps_prochandle *ph, psaddr_t addr, const void *buf, size_t size); ps_err_e ps_pstop(struct ps_prochandle *ph); ps_err_e ps_pcontinue(struct ps_prochandle *ph); ps_err_e ps_lstop(struct ps_prochandle *ph, lwpid_t lwpid); ps_err_e ps_lcontinue(struct ps_prochandle *ph, lwpid_t lwpid); ps_err_e ps_lgetregs(struct ps_prochandle *ph, lwpid_t lwpid, pgregset_t gregset); ps_err_e ps_lsetregs(struct ps_prochandle *ph, lwpid_t lwpid, const pgregset_t gregset); ps_err_e ps_lgetfpregs(struct ps_prochandle *ph, lwpid_t lwpid, prfpregset_t *fpregset); ps_err_e ps_lsetfpregs(struct ps_prochandle *ph, lwpid_t lwpid, const prfpregset_t *fpregset); ps_err_e ps_pauxv(struct ps_prochandle *ph, const auxv_t **auxp); ps_err_e ps_kill(struct ps_prochandle *ph, int sig); ps_err_e ps_lrolltoaddr(struct ps_prochandle *ph, lwpid_t lwpid, psaddr_t go_addr, psaddr_t stop_addr); </pre> |

```
void ps_plog(const char *fmt);
```

SPARC

```
ps_err_e ps_lgetxregsize(struct ps_prochandle *ph, lwpid_t lwpid, int *xregsize);
```

```
ps_err_e ps_lgetxregs(struct ps_prochandle *ph, lwpid_t lwpid, caddr_t xregset);
```

```
ps_err_e ps_lsetxregs(struct ps_prochandle *ph, lwpid_t lwpid, caddr_t xregset);
```

x86

```
ps_err_e ps_lgetLDT(struct ps_prochandle *ph, lwpid_t lwpid, struct ssd *ldt);
```

DESCRIPTION

Every program that links `libthread_db` or `librtld_db` must provide a set of process control primitives that will allow `libthread_db` and `librtld_db` to access memory and registers in the target process, to start and to stop the target process, and to look up symbols in the target process. See `libthread_db(3T)`. For information on `librtld_db`, refer to the *Linker and Libraries Guide*

Refer to the individual reference manual pages that describe these routines for a functional specification that clients of `libthread_db` and `librtld_db` can use to implement this required interface. `<proc_service.h>` lists the C declarations of these routines

FUNCTIONS

| Name | Description |
|----------------------------|---|
| ps_pdmodel() | Returns the data model of the target process. |
| ps_pglobal_lookup() | Looks up the symbol in the symbol table of the load object in the target process and returns its address. |
| ps_pglobal_sym() | Looks up the symbol in the symbol table of the load object in the target process and returns its symbol table entry. |
| ps_pread() | Copies <code>size</code> bytes from the target process to the controlling process. |
| ps_pwrite() | Copies <code>size</code> bytes from the controlling process to the target process. ps_pdread() Identical to ps_pread() . |
| ps_pdwrite() | Identical to ps_pwrite() . |
| ps_ptread() | Identical to ps_pread() . |

| | | |
|-------|--------------------------|---|
| | ps_ptwrite() | Identical to ps_pwrite() . |
| | ps_pstop() | Stops the target process. |
| | ps_pcontinue() | Resumes target process. |
| | ps_lstop() | Stops a single lightweight process (LWP) within the target process. |
| | ps_lcontinue() | Resumes a single LWP within the target process. |
| | ps_lgetregs() | Gets the general registers of the LWP. |
| | ps_lsetregs() | Sets the general registers of the LWP. |
| | ps_lgetfpregs() | Gets the LWP's floating point register set. |
| | ps_lsetfpregs() | Sets the LWP's floating point register set. |
| | ps_pauxv() | Returns a pointer to a read-only copy of the target process's auxiliary vector. |
| | ps_kill() | Sends signal to target process. |
| | ps_lrolltoaddr() | Rolls the LWP out of a critical section when the process is stopped. |
| | ps_plog() | Logs a message. |
| SPARC | ps_lgetxregsize() | Returns the size of the architecture-dependent extra state registers. |
| | ps_lgetxregs() | Gets the extra state registers of the LWP. |
| | ps_lsetxregs() | Sets the extra state registers of the LWP. |
| x86 | ps_lgetLDT() | Reads the local descriptor table of the LWP. |

ATTRIBUTES

See `attributes(5)` for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`libthread_db(3T)`, `attributes(5)`

Linker and Libraries Guide

| | |
|--------------------|--|
| NAME | psignal, sys_siglist – system signal messages |
| SYNOPSIS | <pre> /usr/ucb/cc [flag ...] file ... void psignal(sig, s); unsigned sig ; char * s ; char *sys_siglist[]; </pre> |
| DESCRIPTION | <p>psignal() produces a short message on the standard error file describing the indicated signal. First the argument string <i>s</i> is printed, then a colon, then the name of the signal and a NEWLINE. Most usefully, the argument string is the name of the program which incurred the signal. The signal number should be from among those found in <code><signal.h></code>.</p> <p>To simplify variant formatting of signal names, the vector of message strings <code>sys_siglist</code> is provided; the signal number can be used as an index in this table to get the signal name without the newline. The define <code>NSIG</code> defined in <code><signal.h></code> is the number of messages provided for in the table; it should be checked because new signals may be added to the system before they are added to the table.</p> |
| SEE ALSO | <code>perror(3C)</code> , <code>signal(3C)</code> |
| NOTES | Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported. |

NAME psignal, psigninfo – system signal messages

SYNOPSIS #include <signinfo.h>

```
void psignal(int sig, const char * s);
void psigninfo(signinfo_t * pinfo, char * s);
```

DESCRIPTION The **psignal()** and **psigninfo()** functions produce messages on the standard error output describing a signal. The *sig* argument is a signal that may have been passed as the first argument to a signal handler. The *pinfo* argument is a pointer to a *signinfo* structure that may have been passed as the second argument to an enhanced signal handler. See **sigaction(2)**. The argument string *s* is printed first, followed by a colon and a blank, followed by the message and a NEWLINE character.

USAGE If the application is linked with `-lint1`, then messages printed from these functions are in the native language specified by the LC_MESSAGES locale category. See **setlocale(3C)**.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO **sigaction(2)**, **gettext(3C)**, **perror(3C)**, **setlocale(3C)**, **attributes(5)**, **signinfo(5)**, **signal(5)**

| | |
|----------------------|---|
| NAME | ps_lgetregs, ps_lsetregs, ps_lgetfpregs, ps_lsetfpregs, ps_lgetxregsize, ps_lgetxregs, ps_lsetxregs – routines that access the target process register in libthread_db |
| SYNOPSIS | <pre>#include <proc_service.h> ps_err_e ps_lgetregs(struct ps_prochandle * ph, lwpid_t lid, pgregset_t gregset); ps_err_e ps_lsetregs(struct ps_prochandle * ph, lwpid_t lid, static pgregset_t gregset); ps_err_e ps_lgetfpregs(struct ps_prochandle * ph, lwpid_t lid, prfpregset_t * fpregs); ps_err_e ps_lsetfpregs(struct ps_prochandle * ph, lwpid_t lid, static prfpregset_t * fpregs); ps_err_e ps_lgetxregsize(struct ps_prochandle * ph, lwpid_t lid, int * xregsize); ps_err_e ps_lgetxregs(struct ps_prochandle * ph, lwpid_t lid, caddr_t xregset); ps_err_e ps_lsetxregs(struct ps_prochandle * ph, lwpid_t lid, caddr_t xregset);</pre> |
| DESCRIPTION | <p>ps_lgetregs() , ps_lsetregs() , ps_lgetfpregs() , ps_lsetfpregs() , ps_lgetxregsize() , ps_lgetxregs() , ps_lsetxregs() read and write register sets from lightweight processes (LWP s) within the target process identified by <i>ph</i> . ps_lgetregs() gets the general registers of the LWP identified by <i>lid</i> , and ps_lsetregs() sets them. ps_lgetfpregs() gets the LWP 's floating point register set, while ps_lsetfpregs() sets it.</p> |
| SPARC Only | <p>ps_lgetxregsize() , ps_lgetxregs() , and ps_lsetxregs() are SPARC-specific. They do not need to be defined by a controlling process on non-SPARC architecture. ps_lgetxregsize() returns in * <i>xregsize</i> the size of the architecture-dependent extra state registers. ps_lgetxregs() gets the extra state registers, and ps_lsetxregs() sets them.</p> |
| RETURN VALUES | <p>PS_OK The call returned successfully.</p> <p>PS_NOFPREGS Floating point registers are neither available for this architecture nor for this process.</p> <p>PS_NOXREGS Extra state registers are not available on this architecture.</p> <p>PS_ERR The function did not return successfully.</p> |
| ATTRIBUTES | See attributes(5) for description of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`libthread(3T)`, `libthread_db(3T)`, `proc_service(3T)`,
`libthread_db(4)`, `attributes(5)`

NAME | `ps_pglobal_lookup`, `ps_pglobal_sym` – look up a symbol in the symbol table of the load object in the target process

SYNOPSIS | `#include <proc_service.h>`

```
ps_err_e ps_pglobal_lookup(struct ps_prochandle * ph, const char * object_name,
const char * sym_name, ps_addr_t * sym_addr);

ps_err_e ps_pglobal_sym(struct ps_prochandle * ph, const char * object_name, const
char * sym_name, ps_sym_t * sym);
```

DESCRIPTION | `ps_pglobal_lookup()` looks up the symbol `sym_name` in the symbol table of the load object `object_name` in the target process identified by `ph`. It returns the symbol's value as an address in the target process in `* sym_addr`.

`ps_pglobal_sym()` looks up the symbol `sym_name` in the symbol table of the load object `object_name` in the target process identified by `ph`. It returns the symbol table entry in `* sym`. The value in the symbol table entry is the symbol's value as an address in the target process.

RETURN VALUES

| | |
|-----------------------|---|
| <code>PS_OK</code> | The call completed successfully. |
| <code>PS_NOSYM</code> | The specified symbol was not found. |
| <code>PS_ERR</code> | The function did not return successfully. |

ATTRIBUTES | See `attributes(5)` for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO | `kill(2)`, `libthread(3T)`, `libthread_db(3T)`, `proc_service(3T)`, `libthread_db(4)`, `attributes(5)`

| NAME | ps_pread, ps_pwrite, ps_phread, ps_pdwwrite, ps_pthead, ps_ptwrite – interfaces in libthread_db that target process memory access | | | | |
|----------------------|--|----------------|-----------------|----------|------|
| SYNOPSIS | <pre>#include <proc_service.h> ps_err_e ps_pread(struct ps_prochandle * ph, psaddr_t addr, void * buf, size_t size); ps_err_e ps_pwrite(struct ps_prochandle * ph, psaddr_t addr, const void * buf, size_t size); ps_err_e ps_phread(struct ps_prochandle * ph, psaddr_t addr, void * buf, size_t size); ps_err_e ps_pdwwrite(struct ps_prochandle * ph, psaddr_t addr, const void * buf, size_t size); ps_err_e ps_pthead(struct ps_prochandle * ph, psaddr_t addr, void * buf, size_t size); ps_err_e ps_ptwrite(struct ps_prochandle * ph, psaddr_t addr, const void * buf, size_t size);</pre> | | | | |
| DESCRIPTION | <p>These routines copy data between the target process's address space and the controlling process. ps_pread() copies <i>size</i> bytes from address <i>addr</i> in the target process into <i>buf</i> in the controlling process. ps_pwrite() is like ps_pread() except that the direction of the copy is reversed; data is copied from the controlling process to the target process.</p> <p>ps_phread() and ps_pthead() behave identically to ps_pread() . ps_pdwwrite() and ps_ptwrite() behave identically to ps_pwrite() . These functions can be implemented as simple aliases for the corresponding primary functions. They are artifacts of history that must be maintained.</p> | | | | |
| RETURN VALUES | <p>PS_OK The call returned successfully. <i>size</i> bytes were copied.</p> <p>PS_BADADDR Some part of the address range from <i>addr</i> through <i>addr + size - 1</i> is not part of the target process's address space.</p> <p>PS_ERR The function did not return successfully.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for description of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT Level | Safe | | | | |
| SEE ALSO | libthread(3T) , libthread_db(3T) , proc_service(3T) , libthread_db(4) , attributes(5) | | | | |

| | |
|--------------------|---|
| NAME | ps_pstop, ps_pcontinue, ps_lstop, ps_lcontinue, ps_rolloaddr, ps_kill – process and LWP control in libthread_db |
| SYNOPSIS | <pre>#include <proc_service.h> ps_err_e ps_pstop(struct ps_prochandle * ph); ps_err_e ps_pcontinue(struct ps_prochandle * ph); ps_err_e ps_lstop(struct ps_prochandle * ph, lwpid_t lwpid); ps_err_e ps_lcontinue(struct ps_prochandle * ph, lwpid_t lwpid); ps_err_e ps_rolloaddr(struct ps_prochandle * ph, lwpid_t lwpid, psaddr_t go_addr, psaddr_t stop_addr); ps_err_e ps_kill(struct ps_prochandle * ph, int signum);</pre> |
| DESCRIPTION | <p>ps_pstop() stops the target process identified by <i>ph</i> , while ps_pcontinue() allows it to resume.</p> <p>libthread_db() uses ps_pstop() to freeze the target process while it is under inspection. Within the scope of any single call from outside libthread_db() to a libthread_db() routine, libthread_db() will call ps_pstop() , at most once. If it does, it will call ps_pcontinue() within the scope of the same routine.</p> <p>The controlling process may already have stopped the target process when it calls libthread_db() . In that case, it is not obligated to resume the target process when libthread_db() calls ps_pcontinue() . In other words, ps_pstop() is mandatory, while ps_pcontinue() is advisory. After ps_pstop() , the target process must be stopped; after ps_pcontinue() , the target process may be running.</p> <p>ps_lstop() and ps_lcontinue() stop and resume a single lightweight process (LWP)within the target process <i>ph</i> . They are not currently used by libthread_db() .</p> <p>ps_rolloaddr() is used to roll an LWP forward out of a critical section when the process is stopped. It is also used to run the libthread_db() agent thread on behalf of libthread() . ps_rolloaddr() is always called with the target process stopped, that is, there has been a preceding call to ps_pstop() . The specified LWP must be continued at the address <i>go_addr</i> , or at its current address if <i>go_addr</i> is NULL . It should then be stopped when its execution reaches <i>stop_addr</i> . This routine does not return until the LWP has stopped at <i>stop_addr</i> .</p> <p>ps_kill() directs the signal <i>signum</i> to the target process for which the handle is <i>ph</i> . ps_kill() has the same semantics as kill(2) .</p> |

RETURN VALUES

- PS_OK The call completed successfully. In the case of **ps_pstop()** , the target process is stopped.
- PS_BADLID For **ps_lstop()** , **ps_lcontinue()** and **ps_lrolltoaddr()** ; there is no LWP with id *lwipd* in the target process.
- PS_ERR The function did not return successfully.

ATTRIBUTES

See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

kill(2) , **libthread(3T)** , **libthread_db(3T)** , **proc_service(3T)** , **libthread_db(4)** , **attributes(5)**

| | |
|----------------------|--|
| NAME | pthread_atfork – register fork handlers |
| SYNOPSIS | <pre>cc - mt [<i>flag...</i>] <i>file...</i> -lpthread [<i>library...</i>] #include <sys/types.h> #include <unistd.h> int pthread_atfork(void (*prepare, void),void (*parent, void),void (*child, void));</pre> |
| DESCRIPTION | <p>pthread_atfork() declares fork handlers to be called prior to and following fork(), within the thread that called fork(). The order of calls to pthread_atfork() is important.</p> <p>Before fork() processing begins, the <i>prepare</i> fork handler is called. The <i>prepare</i> handler is not called if its address is <code>NULL</code>.</p> <p>The <i>parent</i> fork handler is called after fork() processing finishes in the parent process, and the <i>child</i> fork handler is called after fork() processing finishes in the child process. If the address of <i>parent</i> or <i>child</i> is <code>NULL</code>, then its handler is not called.</p> <p>The <i>prepare</i> fork handler is called in LIFO (last-in first-out) order, whereas the <i>parent</i> and <i>child</i> fork handlers are called in FIFO (first-in first-out) order. This calling order allows applications to preserve locking order.</p> |
| RETURN VALUES | Upon successful completion, pthread_atfork() returns 0. Otherwise, an error number is returned. |
| ERRORS | <p>ENOMEM Insufficient table space exists to record the fork handler addresses.</p> |
| EXAMPLES | <p>EXAMPLE 1 All multi-threaded applications that call fork() in a POSIX threads program, and which do more than simply call exec() in the child of the fork, should ensure that the child is protected from deadlock.</p> <p>The deadlock scenario: since the "fork-one" model results in cloning only the thread that called fork, it is possible that, at the time of the call, another thread in the parent owns a lock. In the child, this thread is not cloned, and so no thread will unlock this lock in the child. Now, if the single thread in the child needs this lock, there is a deadlock.</p> <p>The problem is more serious with locks in libraries. Since a library writer does not know if the application that is using the library calls fork() or not, the library has to protect itself, for complete correctness, from such a deadlock scenario. If the application that links with this library calls fork() and does not call exec() in the child, and needs a library lock that may be held by some other thread in the parent which is inside the library at the time of the fork,</p> |

then the application deadlocks inside the library. The problem may be solved by using **pthread_atfork()**.

The following is a brief and simple description of how to make a library safe with respect to **fork1()** by using **pthread_atfork()**.

1. Identify all the locks used by the library. Let's say this list is {L1, . . . Ln}. Also identify the locking order for these locks. Let's say that this order is also L1 . . . Ln.
2. Add a call to `pthread_atfork(f1, f2, f3)` in the library's *.init* section. `f1, f2, f3` are defined as follows:

```

f1()
{
    pthread_mutex_lock(L1); |
    pthread_mutex_lock(...); | --> ordered in lock order
    pthread_mutex_lock(Ln); |
}
v

f2()
{
    pthread_mutex_unlock(L1);
    pthread_mutex_unlock(...);
    pthread_mutex_unlock(Ln);
}

f3()
{
    pthread_mutex_unlock(L1);
    pthread_mutex_unlock(...);
    pthread_mutex_unlock(Ln);
}

```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

fork(2), **atexit(3C)**, **attributes(5)**, **standards(5)**

| | |
|----------------------|---|
| NAME | pthread_attr_getdetachstate, pthread_attr_setdetachstate – get or set detachstate attribute |
| SYNOPSIS | <pre>cc - mt [flag ...] file ...- lpthread [library ...] #include <pthread.h> int pthread_attr_setdetachstate(pthread_attr_t * attr, int detachstate); int pthread_attr_getdetachstate(const pthread_attr_t * attr, int * detachstate);</pre> |
| DESCRIPTION | <p>The <i>detachstate</i> attribute controls whether the thread is created in a detached state. If the thread is created detached, then use of the ID of the newly created thread by the pthread_detach() or pthread_join() function is an error.</p> <p>The pthread_attr_setdetachstate() and pthread_attr_getdetachstate() , respectively, set and get the <i>detachstate</i> attribute in the <i>attr</i> object.</p> <p>The <i>detachstate</i> can be set to either PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE . A value of PTHREAD_CREATE_DETACHED causes all threads created with <i>attr</i> to be in the detached state, whereas using a value of PTHREAD_CREATE_JOINABLE causes all threads created with <i>attr</i> to be in the joinable state. The default value of the <i>detachstate</i> attribute is PTHREAD_CREATE_JOINABLE .</p> |
| RETURN VALUES | <p>Upon successful completion, pthread_attr_setdetachstate() and pthread_attr_getdetachstate() return a value of 0 . Otherwise, an error number is returned to indicate the error.</p> <p>The pthread_attr_getdetachstate() function stores the value of the <i>detachstate</i> attribute in <i>detachstate</i> if successful.</p> |
| ERRORS | <p>The pthread_attr_setdetachstate() or pthread_attr_getdetachstate() functions may fail if:</p> <p>EINVAL <i>attr</i> or <i>detachstate</i> is invalid.</p> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`pthread_attr_init(3T)`, `pthread_attr_setstackaddr(3T)`,
`pthread_attr_setstacksize(3T)`, `pthread_create(3T)`,
`attributes(5)`, `standards(5)`

| | |
|--------------------|---|
| NAME | pthread_attr_getguardsize, pthread_attr_setguardsize – get or set the thread guardsize attribute |
| SYNOPSIS | <pre>cc - mt [flag ...] file ...- lpthread [library ...] #include <pthread.h> int pthread_attr_getguardsize(const pthread_attr_t *attr, size_t *guardsize); int pthread_attr_setguardsize(pthread_attr_t *attr, size_t guardsize);</pre> |
| DESCRIPTION | <p>The <i>guardsize</i> attribute controls the size of the guard area for the created thread's stack. The <i>guardsize</i> attribute provides protection against overflow of the stack pointer. If a thread's stack is created with guard protection, the implementation allocates extra memory at the overflow end of the stack as a buffer against stack overflow of the stack pointer. If an application overflows into this buffer an error results (possibly in a SIGSEGV signal being delivered to the thread).</p> <p>The <i>guardsize</i> attribute is provided to the application for two reasons:</p> <ol style="list-style-type: none"> 1. Overflow protection can potentially result in wasted system resources. An application that creates a large number of threads, and which knows its threads will never overflow their stack, can save system resources by turning off guard areas. 2. When threads allocate large data structures on the stack, large guard areas may be needed to detect stack overflow. <p>The pthread_attr_getguardsize() function gets the <i>guardsize</i> attribute in the <i>attr</i> object. This attribute is returned in the <i>guardsize</i> parameter.</p> <p>The pthread_attr_setguardsize() function sets the <i>guardsize</i> attribute in the <i>attr</i> object. The new value of this attribute is obtained from the <i>guardsize</i> parameter. If <i>guardsize</i> is 0 , a guard area will not be provided for threads created with <i>attr</i> . If <i>guardsize</i> is greater than 0 , a guard area of at least size <i>guardsize</i> bytes is provided for each thread created with <i>attr</i> .</p> <p>A conforming implementation is permitted to round up the value contained in <i>guardsize</i> to a multiple of the configurable system variable <code>PAGESIZE</code> . If an implementation rounds up the value of <i>guardsize</i> to a multiple of <code>PAGESIZE</code> , a call to pthread_attr_getguardsize() specifying <i>attr</i> will store in the <i>guardsize</i></p> |

parameter the guard size specified by the previous **pthread_attr_setguardsize()** function call.

The default value of the *guardsize* attribute is `PAGESIZE` bytes. The actual value of `PAGESIZE` is implementation-dependent and may not be the same on all implementations.

If the *stackaddr* attribute has been set (that is, the caller is allocating and managing its own thread stacks), the *guardsize* attribute is ignored and no protection will be provided by the implementation. It is the responsibility of the application to manage stack overflow along with stack allocation and management in this case.

RETURN VALUES

If successful, the **pthread_attr_getguardsize()** and **pthread_attr_setguardsize()** functions return 0 . Otherwise, an error number is returned to indicate the error.

ERRORS

The **pthread_attr_getguardsize()** and **pthread_attr_setguardsize()** functions will fail if:

- EINVAL** The attribute *attr* is invalid.
- EINVAL** The parameter *guardsize* is invalid.
- EINVAL** The parameter *guardsize* contains an invalid value.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

sysconf(3C) , **pthread_attr_init(3T)** , **attributes(5)**

| | | | | | |
|------------------------|---|-----------------------|--|------------------------|--|
| NAME | pthread_attr_getinheritsched, pthread_attr_setinheritsched – get or set inheritsched attribute | | | | |
| SYNOPSIS | <pre>cc - mt [flag ...] file ...- lpthread [library ...] #include <pthread.h> int pthread_attr_setinheritsched(pthread_attr_t * attr, int inheritsched); int pthread_attr_getinheritsched(const pthread_attr_t * attr, int * inheritsched);</pre> | | | | |
| DESCRIPTION | <p>The functions pthread_attr_setinheritsched() and pthread_attr_getinheritsched() , respectively, set and get the <i>inheritsched</i> attribute in the <i>attr</i> argument.</p> <p>When the attribute objects are used by pthread_create() , the <i>inheritsched</i> attribute determines how the other scheduling attributes of the created thread are to be set:</p> <table border="0" style="width: 100%;"> <tr> <td style="vertical-align: top; padding-right: 20px;">PTHREAD_INHERIT_SCHED</td> <td>Specifies that the scheduling policy and associated attributes are to be inherited from the creating thread, and the scheduling attributes in this <i>attr</i> argument are to be ignored.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 20px;">PTHREAD_EXPLICIT_SCHED</td> <td>Specifies that the scheduling policy and associated attributes are to be set to the corresponding values from this attribute object.</td> </tr> </table> <p>The symbols PTHREAD_INHERIT_SCHED and PTHREAD_EXPLICIT_SCHED are defined in the header <pthread.h> .</p> | PTHREAD_INHERIT_SCHED | Specifies that the scheduling policy and associated attributes are to be inherited from the creating thread, and the scheduling attributes in this <i>attr</i> argument are to be ignored. | PTHREAD_EXPLICIT_SCHED | Specifies that the scheduling policy and associated attributes are to be set to the corresponding values from this attribute object. |
| PTHREAD_INHERIT_SCHED | Specifies that the scheduling policy and associated attributes are to be inherited from the creating thread, and the scheduling attributes in this <i>attr</i> argument are to be ignored. | | | | |
| PTHREAD_EXPLICIT_SCHED | Specifies that the scheduling policy and associated attributes are to be set to the corresponding values from this attribute object. | | | | |
| RETURN VALUES | If successful, the pthread_attr_setinheritsched() and pthread_attr_getinheritsched() functions return 0 . Otherwise, an error number is returned to indicate the error. | | | | |
| ERRORS | <p>The pthread_attr_setinheritsched() or pthread_attr_getinheritsched() functions may fail if:</p> <p>EINVAL <i>attr</i> or <i>inheritsched</i> is invalid.</p> | | | | |

USAGE After these attributes have been set, a thread can be created with the specified attributes using **pthread_create()** . Using these routines does not affect the current running thread.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **pthread_attr_init(3T)** , **pthread_attr_setscope(3T)** ,
pthread_attr_setschedpolicy(3T) ,
pthread_attr_setschedparam(3T) , **pthread_create(3T)** ,
pthread_setsched_param(3T) , **attributes(5)** , **standards(5)**

NAME pthread_attr_getschedparam, pthread_attr_setschedparam – get or set schedparam attribute

SYNOPSIS

```
cc - mt [
  flag
  ... ]
file
...- lpthread [
  library
  ... ]

#include <pthread.h>

int pthread_attr_setschedparam(pthread_attr_t * attr, const struct sched_param *
param);

int pthread_attr_getschedparam(const pthread_attr_t * attr, struct sched_param *
param);
```

DESCRIPTION The functions **pthread_attr_setschedparam()** and **pthread_attr_getschedparam()**, respectively, set and get the scheduling parameter attributes in the *attr* argument. The contents of the *param* structure are defined in `<sched.h>`. For the `SCHED_FIFO` and `SCHED_RR` policies, the only required member of *param* is *sched_priority*.

RETURN VALUES If successful, the **pthread_attr_setschedparam()** and **pthread_attr_getschedparam()** functions return 0. Otherwise, an error number is returned to indicate the error.

ERRORS The **pthread_attr_setschedparam()** function may fail if:
EINVAL *attr* is invalid.
The **pthread_attr_getschedparam()** function may fail if:
EINVAL *attr* or *param* is invalid.

USAGE After these attributes have been set, a thread can be created with the specified attributes using **pthread_create()**. Using these routines does not affect the current running thread.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

pthread_attr_init(3T), pthread_attr_setscope(3T),
pthread_attr_setinheritsched(3T),
pthread_attr_setschedpolicy(3T), pthread_create(3T),
pthread_setsched_param(3T), attributes(5), standards(5)

NAME pthread_attr_getschedpolicy, pthread_attr_setschedpolicy – get or set schedpolicy attribute

SYNOPSIS

```
cc - mt [
  flag
  ... ]
file
...- lpthread [
  library
  ... ]

#include <pthread.h>

int pthread_attr_setschedpolicy(pthread_attr_t * attr, int policy);

int pthread_attr_getschedpolicy(const pthread_attr_t * attr, int * policy);
```

DESCRIPTION

The functions **pthread_attr_setschedpolicy()** and **pthread_attr_getschedpolicy()**, respectively, set and get the *schedpolicy* attribute in the *attr* argument.

The supported values of *policy* include `SCHED_FIFO`, `SCHED_RR` and `SCHED_OTHER`, which are defined by the header `<sched.h>`. When threads executing with the scheduling policy `SCHED_FIFO` or `SCHED_RR` are waiting on a mutex, they acquire the mutex in priority order when the mutex is unlocked.

RETURN VALUES

If successful, the **pthread_attr_setschedpolicy()** and **pthread_attr_getschedpolicy()** functions return 0. Otherwise, an error number is returned to indicate the error.

ERRORS

The **pthread_attr_setschedpolicy()** or **pthread_attr_getschedpolicy()** function may fail if:

EINVAL *attr* or *policy* is invalid.

USAGE

After these attributes have been set, a thread can be created with the specified attributes using **pthread_create()**. Using these routines does not affect the current running thread.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

pthread_attr_init(3T), pthread_attr_setscope(3T),
pthread_attr_setinheritsched(3T),
pthread_attr_setschedparam(3T), pthread_create(3T),
pthread_setsched_param(3T), attributes(5), standards(5)

| | |
|----------------------|--|
| NAME | pthread_attr_getscope, pthread_attr_setscope – get or set contention scope attribute |
| SYNOPSIS | <pre>cc - mt [flag ...] file ...- lpthread [library ...] #include <pthread.h> int pthread_attr_setscope(pthread_attr_t * attr, int contention_scope); int pthread_attr_getscope(const pthread_attr_t * attr, int * contention_scope);</pre> |
| DESCRIPTION | <p>The pthread_attr_setscope() and pthread_attr_getscope() functions are used to set and get the <i>contention_scope</i> attribute in the <i>attr</i> object.</p> <p>The pthread_attr_setscope() and pthread_attr_getscope() functions set and get the <i>contention_scope</i> thread attribute in the <i>attr</i> object. The <i>contention_scope</i> value may be set to the following:</p> <p>PTHREAD_SCOPE_SYSTEM Indicates system scheduling contention scope. This thread is permanently "bound" to an LWP, and is also called a bound thread.</p> <p>PTHREAD_SCOPE_PROCESS Indicates process scheduling contention scope. This thread is not "bound" to an LWP, and is also called an unbound thread. PTHREAD_SCOPE_PROCESS, or unbound, is the default.</p> <p>PTHREAD_SCOPE_SYSTEM and PTHREAD_SCOPE_PROCESS are defined by the header <code><pthread.h></code>.</p> |
| RETURN VALUES | If successful, the pthread_attr_setscope() and pthread_attr_getscope() functions return 0. Otherwise, an error number is returned to indicate the error. |
| ERRORS | The pthread_attr_setscope() , or pthread_attr_getscope() , function may fail if: EINVAL <i>attr</i> or <i>contention_scope</i> is invalid. |
| USAGE | After these attributes have been set, a thread can be created with the specified attributes using pthread_create() . Using these routines does not affect the current running thread. |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`pthread_attr_init(3T)`, `pthread_attr_setinheritsched(3T)`,
`pthread_attr_setschedpolicy(3T)`,
`pthread_attr_setschedparam(3T)`, `pthread_create(3T)`,
`pthread_setsched_param(3T)`, `attributes(5)`, `standards(5)`

| NAME | pthread_attr_getstackaddr, pthread_attr_setstackaddr – get or set stackaddr attribute | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc - mt [flag ...] file ...- lpthread [library ...] #include <pthread.h> int pthread_attr_setstackaddr(pthread_attr_t * attr, void * stackaddr); int pthread_attr_getstackaddr(const pthread_attr_t * attr, void ** stackaddr);</pre> | | | | |
| DESCRIPTION | <p>The functions pthread_attr_setstackaddr() and pthread_attr_getstackaddr() , respectively, set and get the thread creation <i>stackaddr</i> attribute in the <i>attr</i> object. The <i>stackaddr</i> default is <code>NULL</code> . See pthread_create(3T) .</p> <p>The <i>stackaddr</i> attribute specifies the location of storage to be used for the created thread's stack. The size of the storage is at least <code>PTHREAD_STACK_MIN</code> .</p> | | | | |
| RETURN VALUES | <p>Upon successful completion, pthread_attr_setstackaddr() and pthread_attr_getstackaddr() return a value of 0 . Otherwise, an error number is returned to indicate the error.</p> <p>If successful, the pthread_attr_getstackaddr() function stores the <i>stackaddr</i> attribute value in <i>stackaddr</i> .</p> | | | | |
| ERRORS | <p>The pthread_attr_setstackaddr() function may fail if:</p> <p>EINVAL <i>attr</i> is invalid.</p> <p>The pthread_attr_getstackaddr() function may fail if:</p> <p>EINVAL <i>attr</i> or <i>stackaddr</i> is invalid.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | <p>pthread_attr_init(3T) , pthread_attr_setdetachstate(3T) , pthread_attr_setstacksize(3T) , pthread_create(3T) , attributes(5) , standards(5)</p> | | | | |

| NAME | pthread_attr_getstacksize, pthread_attr_setstacksize – get or set stacksize attribute | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc - mt [flag ...] file ...- lpthread [library ...] #include <pthread.h> int pthread_attr_setstacksize(pthread_attr_t * attr, size_t stacksize); int pthread_attr_getstacksize(const pthread_attr_t * attr, size_t * stacksize);</pre> | | | | |
| DESCRIPTION | <p>The functions pthread_attr_setstacksize() and pthread_attr_getstacksize() , respectively, set and get the thread creation <i>stacksize</i> attribute in the <i>attr</i> object.</p> <p>The <i>stacksize</i> attribute defines the minimum stack size (in bytes) allocated for the created threads stack. When the <i>stacksize</i> argument is <code>NULL</code> , the default stack size becomes 1 megabyte for 32-bit processes and 2 megabytes for 64-bit processes.</p> | | | | |
| RETURN VALUES | <p>Upon successful completion, pthread_attr_setstacksize() and pthread_attr_getstacksize() return a value of 0 . Otherwise, an error number is returned to indicate the error. The pthread_attr_getstacksize() function stores the <i>stacksize</i> attribute value in <i>stacksize</i> if successful.</p> | | | | |
| ERRORS | <p>The pthread_attr_setstacksize() or pthread_attr_getstacksize() function may fail if:</p> <p>EINVAL <i>attr</i> or <i>stacksize</i> is invalid.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | <p>pthread_attr_init(3T) , pthread_attr_setstackaddr(3T) , pthread_attr_setdetachstate(3T) , pthread_create(3T) , attributes(5) , standards(5)</p> | | | | |

NAME pthread_attr_init, pthread_attr_destroy – initialize or destroy threads attribute object

SYNOPSIS

```
cc - mt [
  flag
  ... ]
file
...- lpthread [
  library
  ... ]

#include <pthread.h>

int pthread_attr_init(pthread_attr_t * attr);

int pthread_attr_destroy(pthread_attr_t * attr);
```

DESCRIPTION

The function **pthread_attr_init()** initializes a thread attributes object *attr* with the default value for all of the individual attributes used by a given implementation.

The resulting attribute object (possibly modified by setting individual attribute values), when used by **pthread_create()**, defines the attributes of the thread created. A single attributes object can be used in multiple simultaneous calls to **pthread_create()**.

The **pthread_attr_init()** function initializes a thread attributes object (*attr*) with the default value for each attribute as follows:

| Attribute | Default Value | Meaning of Default |
|------------------------|-------------------------|-------------------------------------|
| <i>contentionscope</i> | PTHREAD_SCOPE_PROCESS | resource competition within process |
| <i>detachstate</i> | PTHREAD_CREATE_JOINABLE | joinable by other threads |
| <i>stackaddr</i> | NULL | stack allocated by system |
| <i>stacksize</i> | NULL | 1 or 2 megabyte |
| <i>priority</i> | 0 | priority of the thread |
| <i>policy</i> | SCHED_OTHER | determined by system |

| Attribute | Default Value | Meaning of Default |
|---------------------|------------------------|---|
| <i>inheritsched</i> | PTHREAD_EXPLICIT_SCHED | scheduling policy and parameters not inherited but explicitly defined by the attribute object |
| <i>guardsize</i> | PAGESIZE | size of guard area for a thread's created stack |

The **pthread_attr_destroy()** function destroys a thread attributes object (*attr*), which cannot be reused until it is reinitialized. An implementation may cause **pthread_attr_destroy()** to set *attr* to an implementation-dependent invalid value. The behavior of using the attribute after it has been destroyed is undefined.

RETURN VALUES

Upon successful completion, **pthread_attr_init()** and **pthread_attr_destroy()** return a value of 0 . Otherwise, an error number is returned to indicate the error.

ERRORS

The **pthread_attr_init()** function will fail if:

ENOMEM Insufficient memory exists to initialize the thread attributes object.

The **pthread_attr_destroy()** function may fail if:

EINVAL *attr* is invalid.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

sysconf(3C) , **pthread_attr_getdetachstate(3T)** ,
pthread_attr_getguardsize(3T) ,
pthread_attr_getinheritsched(3T) ,
pthread_attr_getschedparam(3T) ,
pthread_attr_getschedpolicy(3T) , **pthread_attr_getscope(3T)** ,
pthread_attr_getstackaddr(3T) , **pthread_attr_getstacksize(3T)** ,
pthread_attr_setdetachstate(3T) ,
pthread_attr_setguardsize(3T) ,
pthread_attr_setinheritsched(3T) ,
pthread_attr_setschedparam(3T) ,
pthread_attr_setschedpolicy(3T) , **pthread_attr_setscope(3T)** ,

```
pthread_attr_setstackaddr(3T), pthread_attr_setstacksize(3T),  
pthread_create(3T), attributes(5), standards(5)
```

| NAME | pthread_cancel – cancel execution of a thread | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc - mt [<i>flag...</i>] <i>file...</i>- lpthread [<i>library...</i>] #include <pthread.h> int pthread_cancel(pthread_t <i>target_thread</i>);</pre> | | | | |
| DESCRIPTION | <p>The pthread_cancel() function requests that <i>target_thread</i> be canceled.</p> <p>By default, cancellation is deferred until <i>target_thread</i> reaches a cancellation point. See cancellation(3T).</p> <p>Cancellation cleanup handlers for <i>target_thread</i> are called when the cancellation is acted on. Upon return of the last cancellation cleanup handler, the thread-specific data destructor functions are called for <i>target_thread</i>. <i>target_thread</i> is terminated when the last destructor function returns.</p> <p>The cancellation processing in <i>target_thread</i> runs asynchronously with respect to the calling thread returning from pthread_cancel().</p> | | | | |
| RETURN VALUES | If successful, the pthread_cancel() function returns 0. Otherwise, an error number is returned to indicate the error. | | | | |
| ERRORS | <p>The pthread_cancel() function may fail if:</p> <p>ESRCH No thread was found with an ID corresponding to that specified by the given thread ID, <i>target_thread</i>.</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | cancellation(3T) , condition(3T) , pthread_cleanup_pop(3T) , pthread_cleanup_push(3T) , pthread_cond_wait(3T) , pthread_cond_timedwait(3T) , pthread_exit(3T) , pthread_join(3T) , pthread_setcancelstate(3T) , pthread_setcanceltype(3T) , pthread_testcancel(3T) , setjmp(3C) , attributes(5) | | | | |
| NOTES | See cancellation(3T) for a discussion of cancellation concepts. | | | | |

NAME pthread_cleanup_pop – pop a thread cancellation cleanup handler

SYNOPSIS

```
cc - mt [ flag... ] file...- lpthread [ library... ]

#include <pthread.h>

void pthread_cleanup_pop(intexecute);
```

DESCRIPTION

pthread_cleanup_pop() removes the cleanup handler routine at the top of the cancellation cleanup stack of the calling thread and executes it if *execute* is non-zero.

When the thread calls **pthread_cleanup_pop()** with a non-zero *execute* argument, the argument at the top of the stack is popped and executed. An argument of 0 pops the handler without executing it.

The Solaris system generates a compile time error if **pthread_cleanup_push()** does not have a matching `pthread_cleanup_pop()`.

Be aware that using **longjmp()** or **siglongjmp()** to jump into or out of a push/pop pair can lead to trouble, as either the matching push or the matching pop statement might not get executed.

RETURN VALUES The **pthread_cleanup_pop()** function returns no value.

ERRORS No errors are defined.

The **pthread_cleanup_pop()** function will not return an error code of EINTR.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **cancellation(3T)**, **condition(3T)**, **pthread_cancel(3T)**, **pthread_cleanup_push(3T)**, **pthread_exit(3T)**, **pthread_join(3T)**, **pthread_setcancelstate(3T)**, **pthread_setcanceltype(3T)**, **pthread_testcancel(3T)**, **setjmp(3C)**, **attributes(5)**

NOTES See **cancellation(3T)** for a discussion of cancellation concepts.

| NAME | pthread_cleanup_push – push a thread cancellation cleanup handler | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc - mt [<i>flag...</i>] <i>file...</i>- lpthread [<i>library...</i>] #include <pthread.h> void pthread_cleanup_push(void (*<i>handler</i>), void *<i>arg</i>);</pre> | | | | |
| DESCRIPTION | <p>pthread_cleanup_push() pushes the specified cancellation cleanup handler routine, <i>handler</i>, onto the cancellation cleanup stack of the calling thread.</p> <p>When a thread exits or is canceled and its cancellation cleanup stack is not empty, the cleanup handlers are invoked with the argument <i>arg</i> in last in, first out (LIFO) order from the cancellation cleanup stack.</p> <p>The Solaris system generates a compile time error if pthread_cleanup_push() does not have a matching pthread_cleanup_pop().</p> <p>Be aware that using longjmp() or siglongjmp() to jump into or out of a push/pop pair can lead to trouble, as either the matching push or the matching pop statement might not get executed.</p> | | | | |
| RETURN VALUES | The pthread_cleanup_push() function returns no value. | | | | |
| ERRORS | <p>No errors are defined.</p> <p>The pthread_cleanup_push() function will not return an error code of <code>EINTR</code>.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" data-bbox="500 1228 1396 1314"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | cancellation(3T) , condition(3T) , longjmp(3C) , pthread_cancel(3T) , pthread_cleanup_pop(3T) , pthread_exit(3T) , pthread_join(3T) , pthread_setcancelstate(3T) , pthread_setcanceltype(3T) , pthread_testcancel(3T) , attributes(5) | | | | |
| NOTES | See cancellation(3T) for a discussion of cancellation concepts. | | | | |

| | |
|----------------------|---|
| NAME | pthread_condattr_getpshared, pthread_condattr_setpshared – get or set the process-shared condition variable attributes |
| SYNOPSIS | <pre>cc - mt [flag ...] file ...- lpthread [library ...] #include <pthread.h> int pthread_condattr_getpshared(const pthread_condattr_t * attr, int * pshared); int pthread_condattr_setpshared(pthread_condattr_t * attr, int pshared);</pre> |
| DESCRIPTION | <p>The pthread_condattr_getpshared() function obtains the value of the <i>process-shared</i> attribute from the attributes object referenced by <i>attr</i> . The pthread_condattr_setpshared() function is used to set the <i>process-shared</i> attribute in an initialized attributes object referenced by <i>attr</i> .</p> <p>The <i>process-shared</i> attribute is set to PTHREAD_PROCESS_SHARED to permit a condition variable to be operated upon by any thread that has access to the memory where the condition variable is allocated, even if the condition variable is allocated in memory that is shared by multiple processes. If the <i>process-shared</i> attribute is PTHREAD_PROCESS_PRIVATE , the condition variable will only be operated upon by threads created within the same process as the thread that initialized the condition variable; if threads of differing processes attempt to operate on such a condition variable, the behavior is undefined. The default value of the attribute is PTHREAD_PROCESS_PRIVATE .</p> <p>Additional attributes, their default values, and the names of the associated functions to get and set those attribute values are implementation-dependent.</p> |
| RETURN VALUES | <p>If successful, the pthread_condattr_setpshared() function returns 0 . Otherwise, an error number is returned to indicate the error.</p> <p>If successful, the pthread_condattr_getpshared() function returns 0 and stores the value of the <i>process-shared</i> attribute of <i>attr</i> into the object referenced by the <i>pshared</i> parameter. Otherwise, an error number is returned to indicate the error.</p> |
| ERRORS | <p>The pthread_condattr_getpshared() and pthread_condattr_setpshared() functions may fail if:</p> <p>EINVAL The value specified by <i>attr</i> is invalid. The pthread_condattr_setpshared() function will fail if:</p> |

EINVAL The new value specified for the attribute is outside the range of legal values for that attribute.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`pthread_condattr_init(3T)`, `pthread_create(3T)`,
`pthread_mutex_init(3T)`, `pthread_cond_init(3T)`, `attributes(5)`

| | |
|----------------------|---|
| NAME | pthread_condattr_init, pthread_condattr_destroy – initialize or destroy condition variable attributes object |
| SYNOPSIS | <pre>cc - mt [flag ...] file ...- lpthread [library ...] #include <pthread.h> int pthread_condattr_init(pthread_condattr_t * attr); int pthread_condattr_destroy(pthread_condattr_t * attr);</pre> |
| DESCRIPTION | <p>The function pthread_condattr_init() initializes a condition variable attributes object <i>attr</i> with the default value for all of the attributes defined by the implementation.</p> <p>At present, the only attribute available is the scope of condition variables. The default scope of the attribute is <code>PTHREAD_PROCESS_PRIVATE</code>.</p> <p>Attempts to initialize previously initialized condition variable attributes object will leave the storage allocated by the previous initialization unallocated.</p> <p>After a condition variable attributes object has been used to initialize one or more condition variables, any function affecting the attributes object (including destruction) does not affect any previously initialized condition variables.</p> <p>The pthread_condattr_destroy() function destroys a condition variable attributes object; the object becomes, in effect, uninitialized. An implementation may cause pthread_condattr_destroy() to set the object referenced by <i>attr</i> to an invalid value. A destroyed condition variable attributes object can be re-initialized using pthread_condattr_init(); the results of otherwise referencing the object after it has been destroyed are undefined.</p> <p>Additional attributes, their default values, and the names of the associated functions to get and set those attribute values are implementation-dependent.</p> |
| RETURN VALUES | If successful, the pthread_condattr_init() and pthread_condattr_destroy() functions return 0. Otherwise, an error number is returned to indicate the error. |
| ERRORS | The pthread_condattr_init() function will fail if: |

ENOMEM Insufficient memory exists to initialize the condition variable attributes object.

The **pthread_condattr_destroy()** function may fail if:

EINVAL The value specified by *attr* is invalid.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

pthread_condattr_getpshared(3T) ,
pthread_condattr_setpshared(3T) , **pthread_cond_init(3T)** ,
pthread_create(3T) , **pthread_mutex_init(3T)** , **attributes(5)**

| | |
|--------------------|---|
| NAME | pthread_cond_init, pthread_cond_destroy – initialize or destroy condition variables |
| SYNOPSIS | <pre>cc - mt [flag ...] file ...- lpthread [library ...] #include <pthread.h> int pthread_cond_init(pthread_cond_t * cond, const pthread_condattr_t * attr); int pthread_cond_destroy(pthread_cond_t * cond); pthread_cond_t cond = PTHREAD_COND_INITIALIZER;</pre> |
| DESCRIPTION | <p>The function pthread_cond_init() initializes the condition variable referenced by <i>cond</i> with attributes referenced by <i>attr</i> . If <i>attr</i> is NULL, the default condition variable attributes are used; the effect is the same as passing the address of a default condition variable attributes object. See pthread_condattr_init(3T) . Upon successful initialization, the state of the condition variable becomes initialized.</p> <p>Attempting to initialize an already initialized. condition variable results in undefined behavior.</p> <p>The function pthread_cond_destroy() destroys the given condition variable specified by <i>cond</i> ; the object becomes, in effect, uninitialized. An implementation may cause pthread_cond_destroy() to set the object referenced by <i>cond</i> to an invalid value. A destroyed condition variable object can be re-initialized using pthread_cond_init() ; the results of otherwise referencing the object after it has been destroyed are undefined.</p> <p>It is safe to destroy an initialized condition variable upon which no threads are currently blocked. Attempting to destroy a condition variable upon which other threads are currently blocked results in undefined behavior.</p> <p>In cases where default condition variable attributes are appropriate, the macro <code>PTHREAD_COND_INITIALIZER</code> can be used to initialize condition variables that are statically allocated. The effect is equivalent to dynamic initialization by a call to pthread_cond_init() with parameter <i>attr</i> specified as NULL, except that no error checks are performed.</p> |

RETURN VALUES

If successful, the **pthread_cond_init()** and **pthread_cond_destroy()** functions return 0 . Otherwise, an error number is returned to indicate the error. The EBUSY and EINVAL error checks, if implemented, act as if they were performed immediately at the beginning of processing for the function and caused an error return prior to modifying the state of the condition variable specified by *cond* .

ERRORS

The **pthread_cond_init()** function will fail if:

- EAGAIN** The system lacked the necessary resources (other than memory) to initialize another condition variable.
- ENOMEM** Insufficient memory exists to initialize the condition variable.
- The **pthread_cond_init()** function may fail if:
- EBUSY** The implementation has detected an attempt to re-initialize the object referenced by *cond* , a previously initialized, but not yet destroyed, condition variable.
- EINVAL** The value specified by *attr* is invalid.

The **pthread_cond_destroy()** function may fail if:

- EBUSY** The implementation has detected an attempt to destroy the object referenced by *cond* while it is referenced (for example, while being used in a **pthread_cond_wait()** or **pthread_cond_timedwait()**) by another thread.
- EINVAL** The value specified by *cond* is invalid.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

condition(3T) , **pthread_cond_signal(3T)** ,
pthread_cond_broadcast(3T) , **pthread_cond_wait(3T)** ,
pthread_cond_timedwait(3T) , **pthread_condattr_init(3T)** ,
attributes(5) , **standards(5)**

| | |
|----------------------|---|
| NAME | pthread_cond_signal, pthread_cond_broadcast – signal or broadcast a condition |
| SYNOPSIS | <pre>cc - mt [<i>flag</i> ...] <i>file</i> ...- lpthread [<i>library</i> ...] #include <pthread.h> int pthread_cond_signal(pthread_cond_t * <i>cond</i>); int pthread_cond_broadcast(pthread_cond_t * <i>cond</i>);</pre> |
| DESCRIPTION | <p>These two functions are used to unblock threads blocked on a condition variable.</p> <p>The pthread_cond_signal() call unblocks at least one of the threads that are blocked on the specified condition variable <i>cond</i> (if any threads are blocked on <i>cond</i>).</p> <p>The pthread_cond_broadcast() call unblocks all threads currently blocked on the specified condition variable <i>cond</i>.</p> <p>If more than one thread is blocked on a condition variable, the scheduling policy determines the order in which threads are unblocked. When each thread unblocked as a result of a pthread_cond_signal() or pthread_cond_broadcast() returns from its call to pthread_cond_wait() or pthread_cond_timedwait(), the thread owns the mutex with which it called pthread_cond_wait() or pthread_cond_timedwait(). The thread(s) that are unblocked contend for the mutex according to the scheduling policy (if applicable), and as if each had called pthread_mutex_lock().</p> <p>The pthread_cond_signal() or pthread_cond_broadcast() functions may be called by a thread whether or not it currently owns the mutex that threads calling pthread_cond_wait() or pthread_cond_timedwait() have associated with the condition variable during their waits; however, if predictable scheduling behavior is required, then that mutex is locked by the thread calling pthread_cond_signal() or pthread_cond_broadcast().</p> <p>The pthread_cond_signal() and pthread_cond_broadcast() functions have no effect if there are no threads currently blocked on <i>cond</i>.</p> |
| RETURN VALUES | <p>If successful, the pthread_cond_signal() and pthread_cond_broadcast() functions return 0. Otherwise, an error number is returned to indicate the error.</p> |

ERRORS The `pthread_cond_signal()` and `pthread_cond_broadcast()` function may fail if:
EINVAL The value `cond` does not refer to an initialized condition variable.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO `condition(3T)`, `pthread_cond_init(3T)`, `pthread_cond_wait(3T)`, `pthread_cond_timedwait(3T)`, `attributes(5)`, `standards(5)`

| | |
|--------------------|--|
| NAME | pthread_cond_wait, pthread_cond_timedwait – wait on a condition |
| SYNOPSIS | <pre> cc - mt [<i>flag</i> ...] <i>file</i> ...- lpthread [<i>library</i> ...] #include <pthread.h> int pthread_cond_wait(pthread_cond_t * <i>cond</i>); int pthread_cond_timedwait(pthread_cond_t * <i>cond</i>, pthread_mutex_t * <i>mutex</i>, const struct timespec * <i>abstime</i>); </pre> |
| DESCRIPTION | <p>The pthread_cond_wait() and pthread_cond_timedwait() functions are used to block on a condition variable. They are called with <i>mutex</i> locked by the calling thread or undefined behaviour will result.</p> <p>These functions atomically release <i>mutex</i> and cause the calling thread to block on the condition variable <i>cond</i>; atomically here means “atomically with respect to access by another thread to the mutex and then the condition variable”. That is, if another thread is able to acquire the mutex after the about-to-block thread has released it, then a subsequent call to pthread_cond_signal() or pthread_cond_broadcast() in that thread behaves as if it were issued after the about-to-block thread has blocked.</p> <p>Upon successful return, the mutex has been locked and is owned by the calling thread.</p> <p>When using condition variables there is always a boolean predicate, an invariant, associated with each condition wait that must be true before the thread should proceed. Spurious wakeups from the pthread_cond_wait() or pthread_cond_timedwait() functions may occur. Since the return from pthread_cond_wait() or pthread_cond_timedwait() does not imply anything about the value of this predicate, the predicate should always be re-evaluated.</p> <p>The order in which blocked threads are awakened by cond_signal() or cond_broadcast() is determined by the scheduling policy. See pthreads(3T).</p> <p>The effect of using more than one mutex for concurrent pthread_cond_wait() or pthread_cond_timedwait() operations on the same condition variable will result in undefined behavior.</p> <p>A condition wait (whether timed or not) is a cancellation point. When the cancelability enable state of a thread is set to <code>PTHREAD_CANCEL_DEFERRED</code>, a</p> |

side effect of acting upon a cancellation request while in a condition wait is that the mutex is re-acquired before calling the first cancellation cleanup handler.

A thread that has been unblocked because it has been canceled while blocked in a call to **pthread_cond_wait()** or **pthread_cond_timedwait()** does not consume any condition signal that may be directed concurrently at the condition variable if there are other threads blocked on the condition variable.

The **pthread_cond_timedwait()** function is the same as **pthread_cond_wait()** except that an error is returned if the absolute time specified by *abstime* passes (that is, system time equals or exceeds *abstime*) before the condition *cond* is signaled or broadcasted, or if the absolute time specified by *abstime* has already been passed at the time of the call. When such time-outs occur, **pthread_cond_timedwait()** will nonetheless release and reacquire the mutex referenced by *mutex*. The function **pthread_cond_timedwait()** is also a cancellation point.

If a signal is delivered to a thread waiting for a condition variable, upon return from the signal handler the thread resumes waiting for the condition variable as if it was not interrupted, or it returns 0 due to spurious wakeup.

RETURN VALUES

Except in the case of ETIMEDOUT, all these error checks act as if they were performed immediately at the beginning of processing for the function and cause an error return, in effect, prior to modifying the state of the mutex specified by *mutex* or the condition variable specified by *cond*.

Upon successful completion, a value of 0 is returned. Otherwise, an error number is returned to indicate the error.

ERRORS

The **pthread_cond_timedwait()** function will fail if:

ETIMEDOUT The time specified by *abstime* to **pthread_cond_timedwait()** has passed.

The **pthread_cond_wait()** and **pthread_cond_timedwait()** functions may fail if:

EINVAL The value specified by *cond*, *mutex*, or *abstime* is invalid.

EINVAL Different mutexes were supplied for concurrent **pthread_cond_wait()** or **pthread_cond_timedwait()** operations on the same condition variable.

EINVAL The mutex was not owned by the current thread at the time of the call.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`condition(3T)`, `pthread_cond_signal(3T)`,
`pthread_cond_broadcast(3T)`, `attributes(5)`, `standards(5)`

| | |
|--------------------|---|
| NAME | pthread_create – create a thread |
| SYNOPSIS | <pre>cc - mt [<i>flag...</i>] <i>file...</i>- lpthread [<i>library...</i>] #include <pthread.h> int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine, void*), void *arg);</pre> |
| DESCRIPTION | <p>The pthread_create() function is used to create a new thread, with attributes specified by <i>attr</i>, within a process. If <i>attr</i> is <code>NULL</code>, the default attributes are used. (See pthread_attr_init(3T)). If the attributes specified by <i>attr</i> are modified later, the thread's attributes are not affected. Upon successful completion, pthread_create() stores the ID of the created thread in the location referenced by <i>thread</i>.</p> <p>The thread is created executing <i>start_routine</i> with <i>arg</i> as its sole argument. If the <i>start_routine</i> returns, the effect is as if there was an implicit call to pthread_exit() using the return value of <i>start_routine</i> as the exit status. Note that the thread in which main() was originally invoked differs from this. When it returns from main(), the effect is as if there was an implicit call to exit() using the return value of main() as the exit status.</p> <p>The signal state of the new thread is initialised as follows:</p> <ul style="list-style-type: none"> ■ The signal mask is inherited from the creating thread. ■ The set of signals pending for the new thread is empty. <p>Default thread creation:</p> <pre>pthread_t tid; void *start_func(void *), *arg; pthread_create(&tid, NULL, start_func, arg);</pre> <p>This would have the same effect as:</p> <pre>pthread_attr_t attr; pthread_attr_init(&attr); /* initialize attr with default attributes */ pthread_create(&tid, &attr, start_func, arg);</pre> <p>User-defined thread creation: To create a thread that is scheduled on a system-wide basis, use:</p> <pre>pthread_attr_init(&attr); /* initialize attr with default attributes */ pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM); /* system-wide contention */</pre> |

```
pthread_create(&tid, &attr, start_func, arg);
```

To customize the attributes for POSIX threads, see `pthread_attr_init(3T)`.

A new thread created with `pthread_create()` uses the stack specified by the `stackaddr` attribute, and the stack continues for the number of bytes specified by the `stacksize` attribute. By default, the stack size is 1 megabyte for 32-bit processes and 2 megabyte for 64-bit processes (see `pthread_attr_setstacksize(3T)`). If the default is used for both the `stackaddr` and `stacksize` attributes, `pthread_create()` creates a stack for the new thread with at least 1 megabyte for 32-bit processes and 2 megabyte for 64-bit processes. (For customizing stack sizes, see NOTES).

If `pthread_create()` fails, no new thread is created and the contents of the location referenced by `thread` are undefined.

RETURN VALUES

If successful, the `pthread_create()` function returns 0. Otherwise, an error number is returned to indicate the error.

ERRORS

The `pthread_create()` function will fail if:

- | | |
|---------------|---|
| ENOMEM | The system lacked the necessary resources to create another thread. |
| EINVAL | The value specified by <code>attr</code> is invalid. |
| EPERM | The caller does not have appropriate permission to set the required scheduling parameters or scheduling policy. |

EXAMPLES

EXAMPLE 1 This is an example of concurrency with multi-threading. Since POSIX threads and Solaris threads are fully compatible even within the same process, this example uses `pthread_create()` if you execute `a.out 0`, or `thr_create()` if you execute `a.out 1`.

Five threads are created that simultaneously perform a time-consuming function, `sleep(10)`. If the execution of this process is timed, the results will show that all five individual calls to `sleep` for ten-seconds completed in about ten seconds, even on a uniprocessor. If a single-threaded process calls `sleep(10)` five times, the execution time will be about 50-seconds.

The command-line to time this process is:

```
/usr/bin/time a.out 0 (for POSIX threading)
```

or

```
/usr/bin/time a.out 1 (for Solaris threading)
```

```
/* cc thisfile.c -lthread -lpthread */
#define _REENTRANT /* basic 3-lines for threads */
```

```

#include <pthread.h>
#include <thread.h>

#define NUM_THREADS 5
#define SLEEP_TIME 10

void *sleeping(void *); /* thread routine */
int i;
thread_t tid[NUM_THREADS]; /* array of thread IDs */

int
main(int argc, char *argv[])
{
    if (argc == 1) {
        printf("use 0 as arg1 to use pthread_create()\n");
        printf("or use 1 as arg1 to use thr_create()\n");
        return (1);
    }

    switch (*argv[1]) {
    case '0': /* POSIX */
        for ( i = 0; i < NUM_THREADS; i++)
            pthread_create(&tid[i], NULL, sleeping,
                (void *)SLEEP_TIME);
        for ( i = 0; i < NUM_THREADS; i++)
            pthread_join(tid[i], NULL);
        break;

    case '1': /* Solaris */
        for ( i = 0; i < NUM_THREADS; i++)
            thr_create(NULL, 0, sleeping, (void *)SLEEP_TIME, 0,
                &tid[i]);
        while (thr_join(NULL, NULL, NULL) == 0)
            ;
        break;
    } /* switch */
    printf("main() reporting that all %d threads have terminated\n", i);
    return (0);
} /* main */

void *
sleeping(void *arg)
{
    int sleep_time = (int)arg;
    printf("thread %d sleeping %d seconds ...\n", thr_self(), sleep_time);
    sleep(sleep_time);
    printf("\nthread %d awakening\n", thr_self());
    return (NULL);
}

```

EXAMPLE 2 If `main()` had not waited for the completion of the other threads (using `pthread_join(3T)` or `thr_join(3T)`), it would have continued to process

concurrently until it reached the end of its routine and the entire process would have exited prematurely (see `exit(2)`).

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`fork(2)`, `sysconf(3C)`, `pthread_attr_init(3T)`, `pthread_cancel(3T)`, `pthread_exit(3T)`, `pthread_join(3T)`, `attributes(5)`, `standards(5)`

NOTES

MT application threads execute independently of each other, thus their relative behavior is unpredictable. Therefore, it is possible for the thread executing `main()` to finish before all other user application threads.

`pthread_join(3T)`, on the other hand, must specify the terminating thread (IDs) for which it will wait.

A user-specified stack size must be greater than the value `PTHREAD_STACK_MIN`. A minimum stack size may not accommodate the stack frame for the user thread function `start_func`. If a stack size is specified, it must accommodate `start_func` requirements and the functions that it may call in turn, in addition to the minimum requirement.

It is usually very difficult to determine the runtime stack requirements for a thread. `PTHREAD_STACK_MIN` specifies how much stack storage is required to execute a NULL `start_func`. The total runtime requirements for stack storage are dependent on the storage required to do runtime linking, the amount of storage required by library runtimes (as `printf()`) that your thread calls. Since these storage parameters are not known before the program runs, it is best to use default stacks. If you know your runtime requirements or decide to use stacks that are larger than the default, then it makes sense to specify your own stacks.

| NAME | pthread_detach – detach a thread | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc - mt [<i>flag...</i>] <i>file...</i>- lpthread [<i>library...</i>] #include <pthread.h> int pthread_detach(pthread_t <i>thread</i>);</pre> | | | | |
| DESCRIPTION | <p>The pthread_detach() function is used to indicate to the implementation that storage for the thread <i>thread</i> can be reclaimed when that thread terminates. In other words, pthread_detach() dynamically resets the <i>detachstate</i> attribute of the thread to <code>PTHREAD_CREATE_DETACHED</code>. After a successful call to this function, it would not be necessary to reclaim the thread using pthread_join(). See pthread_join(3T). If <i>thread</i> has not terminated, pthread_detach() will not cause it to terminate. The effect of multiple pthread_detach() calls on the same target thread is unspecified.</p> | | | | |
| RETURN VALUES | If successful, pthread_detach() returns 0. Otherwise, an error number is returned to indicate the error. | | | | |
| ERRORS | <p>The pthread_detach() function will fail if:</p> <p>EINVAL The implementation has detected that the value specified by <i>thread</i> does not refer to a joinable thread.</p> <p>ESRCH No thread could be found corresponding to that specified by the given thread ID.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | pthread_create(3T) , pthread_join(3T) , attributes(5) , standards(5) | | | | |

NAME pthread_equal – compare thread IDs

SYNOPSIS

```
cc - mt [ flag... ] file... -lpthread [ library... ]

#include <pthread.h>

int pthread_equal(pthread_t t1, pthread_t t2);
```

DESCRIPTION This function compares the thread IDs *t1* and *t2*.

RETURN VALUES The **pthread_equal()** function returns a non-zero value if *t1* and *t2* are equal. Otherwise, 0 is returned.

If *t1* or *t2* is an invalid thread ID, the behavior is undefined.

ERRORS No errors are defined.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **pthread_create(3T)**, **pthread_self(3T)**, **attributes(5)**

NOTES Solaris thread IDs do not require an equivalent function because the `thread_t` structure is an unsigned int.

| | |
|----------------------|---|
| NAME | pthread_exit - terminate calling thread |
| SYNOPSIS | <pre>cc - mt [<i>flag...</i>] <i>file...</i>- lpthread [<i>library...</i>] #include <pthread.h> void pthread_exit(void *<i>value_ptr</i>);</pre> |
| DESCRIPTION | <p>The pthread_exit() function terminates the calling thread, in a similar way that exit(3C) terminates the calling process. If the thread is not detached, the exit status specified by <i>value_ptr</i> is made available to any successful join with the terminating thread. See pthread_join(3T). Any cancellation cleanup handlers that have been pushed and not yet popped are popped in the reverse order that they were pushed and then executed. After all cancellation cleanup handlers have been executed, if the thread has any thread-specific data, appropriate destructor functions will be called in an unspecified order. Thread termination does not release any application visible process resources, including, but not limited to, mutexes and file descriptors, nor does it perform any process level cleanup actions, including, but not limited to, calling any atexit() routines that may exist.</p> <p>An implicit call to pthread_exit() is made when a thread other than the thread in which main() was first invoked returns from the start routine that was used to create it. The function's return value serves as the thread's exit status.</p> <p>The behavior of pthread_exit() is undefined if called from a cancellation cleanup handler or destructor function that was invoked as a result of either an implicit or explicit call to pthread_exit().</p> <p>After a thread has terminated, the result of access to local (auto) variables of the thread is undefined. Thus, references to local variables of the exiting thread should not be used for the pthread_exit() <i>value_ptr</i> parameter value.</p> <p>The process exits with an exit status of 0 after the last thread has been terminated. The behavior is as if the implementation called exit() with a 0 argument at thread termination time.</p> |
| RETURN VALUES | The pthread_exit() function cannot return to its caller. |
| ERRORS | No errors are defined. |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`exit(3C)`, `pthread_cancel(3T)`, `pthread_create(3T)`,
`pthread_join(3T)`, `pthread_key_create(3T)`, `attributes(5)`,
`standards(5)`

| | |
|----------------------|--|
| NAME | pthread_getconcurrency, pthread_setconcurrency – get or set level of concurrency |
| SYNOPSIS | <pre>cc - mt [<i>flag</i> ...] <i>file</i> ...- lpthread [<i>library</i> ...] #include <pthread.h> int pthread_getconcurrency(void);int pthread_setconcurrency(int <i>new_level</i>);</pre> |
| DESCRIPTION | <p>Unbound threads in a process may or may not be required to be simultaneously active. By default, the threads implementation ensures that a sufficient number of threads are active so that the process can continue to make progress. While this conserves system resources, it may not produce the most effective level of concurrency.</p> <p>The pthread_setconcurrency() function allows an application to inform the threads implementation of its desired concurrency level, <i>new_level</i> . The actual level of concurrency provided by the implementation as a result of this function call is unspecified.</p> <p>If <i>new_level</i> is 0 , it causes the implementation to maintain the concurrency level at its discretion as if pthread_setconcurrency() was never called.</p> <p>The pthread_getconcurrency() function returns the value set by a previous call to the pthread_setconcurrency() function. If the pthread_setconcurrency() function was not previously called, this function returns 0 to indicate that the implementation is maintaining the concurrency level.</p> <p>When an application calls pthread_setconcurrency() it is informing the implementation of its desired concurrency level. The implementation uses this as a hint, not a requirement.</p> <p>If an implementation does not support multiplexing of user threads on top of several kernel scheduled entities, the pthread_setconcurrency() and pthread_getconcurrency() functions will be provided for source code compatibility but they will have no effect when called. To maintain the function semantics, the <i>new_level</i> parameter will be saved when pthread_setconcurrency() is called so that a subsequent call to pthread_getconcurrency() returns the same value.</p> |
| RETURN VALUES | If successful, the pthread_setconcurrency() function returns 0 . Otherwise, an error number is returned to indicate the error. |

The **pthread_getconcurrency()** function always returns the concurrency level set by a previous call to **pthread_setconcurrency()** . If the **pthread_setconcurrency()** function has never been called, **pthread_getconcurrency()** returns 0 .

ERRORS

The **pthread_setconcurrency()** function will fail if:

EINVAL The value specified by *new_level* is negative.

EAGAIN The value specific by *new_level* would cause a system resource to be exceeded.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

pthread_create(3T) , **pthread_attr_init(3T)** , **attributes(5)**

| | |
|----------------------|---|
| NAME | pthread_getschedparam, pthread_setschedparam – access dynamic thread scheduling parameters |
| SYNOPSIS | <pre>cc - mt [<i>flag</i> ...] <i>file</i> ...- lpthread [<i>library</i> ...] #include <pthread.h> int pthread_getschedparam(pthread_t <i>thread</i>, int * <i>policy</i>, struct sched_param * <i>param</i>); int pthread_setschedparam(pthread_t <i>thread</i>, jint * <i>policy</i>, const struct sched_param * <i>param</i>);</pre> |
| DESCRIPTION | <p>The pthread_getschedparam() and pthread_setschedparam() allow the scheduling policy and scheduling parameters of individual threads within a multi-threaded process to be retrieved and set. Supported policies are SCHED_FIFO, SCHED_RR, and SCHED_OTHER. See pthreads(3T). For SCHED_FIFO, SCHED_RR, and SCHED_OTHER, the affected scheduling parameter is the <i>sched_priority</i> member of the sched_param structure.</p> <p>The pthread_getschedparam() function retrieves the scheduling policy and scheduling parameters for the thread whose thread ID is given by <i>thread</i> and stores those values in <i>policy</i> and <i>param</i>, respectively. The priority value returned from pthread_getschedparam() is the value specified by the most recent pthread_setschedparam() or pthread_create() call affecting the target thread, and reflects any temporary adjustments to its priority as a result of any priority inheritance or ceiling functions. The pthread_setschedparam() function sets the scheduling policy and associated scheduling parameters for the thread whose thread ID is given by <i>thread</i> to the policy and associated parameters provided in <i>policy</i> and <i>param</i>, respectively.</p> <p>If the pthread_setschedparam() function fails, no scheduling parameters will be changed for the target thread.</p> |
| RETURN VALUES | If successful, the pthread_getschedparam() and pthread_setschedparam() functions return 0. Otherwise, an error number is returned to indicate the error. |
| ERRORS | The pthread_getschedparam() function may fail if: |

ESRCH The value specified by *thread* does not refer to a existing thread.

The **pthread_setschedparam()** function may fail if:

EINVAL The value specified by *policy* or one of the scheduling parameters associated with the scheduling policy *policy* is invalid.

EPERM The caller does not have the appropriate permission to set either the scheduling parameters or the scheduling policy of the specified thread.

ESRCH The value specified by *thread* does not refer to a existing thread.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

pthread_attr_init(3T), **pthreads(3T)**, **sched_setparam(3R)**, **sched_getparam(3R)**, **sched_setscheduler(3R)**, **sched_getscheduler(3R)**, **attributes(5)**, **standards(5)**

| | |
|----------------------|---|
| NAME | pthread_getspecific, pthread_setspecific – manage thread-specific data |
| SYNOPSIS | <pre>cc - mt [<i>flag</i> ...] <i>file</i> ...- lpthread [<i>library</i> ...] #include <pthread.h> int pthread_setspecific(pthread_key_t key, const void * value); void * pthread_getspecific(pthread_key_t key);</pre> |
| DESCRIPTION | <p>The pthread_setspecific() function associates a thread-specific <i>value</i> with a <i>key</i> obtained by way of a previous call to pthread_key_create() . Different threads may bind different values to the same key. These values are typically pointers to blocks of dynamically allocated memory that have been reserved for use by the calling thread.</p> <p>The pthread_getspecific() function returns the value currently bound to the specified <i>key</i> on behalf of the calling thread.</p> <p>The effect of calling pthread_setspecific() or pthread_getspecific() with a <i>key</i> value not obtained from pthread_key_create() or after <i>key</i> has been deleted with pthread_key_delete() is undefined.</p> <p>Both pthread_setspecific() and pthread_getspecific() may be called from a thread-specific data destructor function. However, calling pthread_setspecific() from a destructor may result in lost storage or infinite loops.</p> |
| RETURN VALUES | <p>The pthread_getspecific() function returns the thread-specific data value associated with the given <i>key</i> . If no thread-specific data value is associated with <i>key</i> , then the value <code>NULL</code> is returned.</p> <p>Upon successful completion, the pthread_setspecific() function returns <code>0</code> . Otherwise, an error number is returned to indicate the error.</p> |
| ERRORS | <p>The pthread_setspecific() function will fail if:</p> <p>ENOMEM Insufficient memory exists to associate the value with the key.</p> <p>The pthread_setspecific() function may fail if:</p> <p>EINVAL The key value is invalid.</p> <p>The pthread_getspecific() function does not return errors.</p> |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`pthread_key_create(3T)` `attributes(5)` , `standards(5)`

| | |
|----------------------|--|
| NAME | pthread_join – wait for thread termination |
| SYNOPSIS | <pre>cc - mt [<i>flag...</i>] <i>file...</i>- lpthread [<i>library...</i>] #include <pthread.h> int pthread_join(pthread_t <i>thread</i>, void **<i>value_ptr</i>);</pre> |
| DESCRIPTION | <p>The pthread_join() function suspends processing of the calling thread until the target <i>thread</i> completes. <i>thread</i> must be a member of the current process and it cannot be a detached or daemon thread. See pthread_create(3T).</p> <p>Several threads cannot wait for the same thread to complete; one thread will complete successfully and the others will terminate with an error of ESRCH. pthread_join() will not block processing of the calling thread if the target <i>thread</i> has already terminated.</p> <p>pthread_join() returns successfully when the target <i>thread</i> terminates. If a pthread_join() call returns successfully with a non-null <i>status</i> argument, the value passed to pthread_exit(3T) by the terminating thread will be placed in the location referenced by <i>status</i>.</p> <p>If the pthread_join() calling thread is cancelled, then the target <i>thread</i> will remain joinable by pthread_join(). However, the calling thread may set up a cancellation cleanup handler on <i>thread</i> prior to the join call, which may detach the target thread by calling pthread_detach(3T). (See pthread_detach(3T) and pthread_cancel(3T).)</p> |
| RETURN VALUES | If successful, the pthread_join() function returns 0. Otherwise, an error number is returned to indicate the error. |
| ERRORS | <p>The pthread_join() function will fail if:</p> <p>EINVAL The implementation has detected that the value specified by <i>thread</i> does not refer to a joinable thread.</p> <p>ESRCH No thread could be found corresponding to that specified by the given thread ID.</p> <p>The pthread_join() function may fail if:</p> <p>EDEADLK A recursive deadlock was detected, the value of <i>thread</i> specifies the calling thread.</p> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO `wait(2)`, `pthread_create(3T)`, `attributes(5)`, `standards(5)`

NOTES `pthread_join(3T)`, must specify the *thread* ID for whose termination it will wait.

Calling `pthread_join()` also "detaches" the thread, that is, `pthread_join()` includes the effect of `pthread_detach()`. Hence, if a thread were to be cancelled when blocked in `pthread_join()`, an explicit detach would have to be done in the cancellation cleanup handler. In fact, the routine `pthread_detach()` exists mainly for this reason.

| | |
|----------------------|---|
| NAME | pthread_key_create – create thread-specific data key |
| SYNOPSIS | <pre>cc - mt [<i>flag...</i>] <i>file...</i>- lpthread [<i>library...</i>] #include <pthread.h> int pthread_key_create(pthread_key_t *key, void (*<i>destructor</i>, void*));</pre> |
| DESCRIPTION | <p>This function creates a thread-specific data key visible to all threads in the process. Key values provided by pthread_key_create() are opaque objects used to locate thread-specific data. Although the same key value may be used by different threads, the values bound to the key by pthread_setspecific() are maintained on a per-thread basis and persist for the life of the calling thread.</p> <p>Upon key creation, the value <code>NULL</code> is associated with the new key in all active threads. Upon thread creation, the value <code>NULL</code> is associated with all defined keys in the new thread.</p> <p>An optional destructor function may be associated with each key value. At thread exit, if a key value has a non-<code>NULL</code> destructor pointer, and the thread has a non-<code>NULL</code> value associated with that key, the function pointed to is called with the current associated value as its sole argument. Destructors can be called in any order.</p> <p>If, after all the destructors have been called for all keys with non-<code>NULL</code> values, there are still some keys with non-<code>NULL</code> values, the process will be repeated. If, after at least <code>PTHREAD_DESTRUCTOR_ITERATIONS</code> iterations of destructor calls for outstanding non-<code>NULL</code> values, there are still some keys with non-<code>NULL</code> values, the process is continued, even though this might result in an infinite loop.</p> |
| RETURN VALUES | If successful, the pthread_key_create() function stores the newly created key value at <i>key</i> and returns 0. Otherwise, an error number is returned to indicate the error. |
| ERRORS | <p>The pthread_key_create() function will fail if:</p> <p>EAGAIN The system lacked the necessary resources to create another thread-specific data key, or the system-imposed limit on the total number of keys per process <code>PTHREAD_KEYS_MAX</code> has been exceeded.</p> <p>ENOMEM Insufficient memory exists to create the key.</p> <p>The pthread_key_create() function will not return an error code of <code>EINTR</code>.</p> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`pthread_getspecific(3T)`, `pthread_setspecific(3T)`,
`pthread_key_delete(3T)`, `attributes(5)`, `standards(5)`

| NAME | pthread_key_delete – delete thread-specific data key | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc - mt [<i>flag...</i>] <i>file...</i>- lpthread [<i>library...</i>] #include <pthread.h> int pthread_key_delete(pthread_key_t key);</pre> | | | | |
| DESCRIPTION | <p>This function deletes a thread-specific data key previously returned by pthread_key_create(). The thread-specific data values associated with <i>key</i> need not be <code>NULL</code> at the time pthread_key_delete() is called. It is the responsibility of the application to free any application storage or perform any cleanup actions for data structures related to the deleted key or associated thread-specific data in any threads; this cleanup can be done either before or after pthread_key_delete() is called. Any attempt to use <i>key</i> following the call to pthread_key_delete() results in undefined behaviour.</p> <p>The pthread_key_delete() function is callable from within destructor functions. No destructor functions will be invoked by pthread_key_delete(). Any destructor function that may have been associated with <i>key</i> will no longer be called upon thread exit.</p> | | | | |
| RETURN VALUES | If successful, the pthread_key_delete() function returns 0. Otherwise, an error number is returned to indicate the error. | | | | |
| ERRORS | <p>The pthread_key_delete() function may fail if:</p> <p>EINVAL The <i>key</i> value is invalid.</p> <p>The pthread_key_delete() function will not return an error code of <code>EINTR</code>.</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | pthread_key_create(3T) , attributes(5) , standards(5) | | | | |

NAME pthread_kill – send a signal to a thread

SYNOPSIS cc - mt [*flag...*] *file...*-lpthread [*library...*]

```
#include <signal.h>
#include <pthread.h>

int pthread_kill(pthread_t thread, int sig);
```

DESCRIPTION The **pthread_kill()** function is used to request that a signal be delivered to the specified thread.

As in **kill()**, if *sig* is 0, error checking is performed but no signal is actually sent.

RETURN VALUES Upon successful completion, the function returns a value of 0. Otherwise the function returns an error number. If the **pthread_kill()** function fails, no signal is sent.

ERRORS The **pthread_kill()** function will fail if:

ESRCH No thread could be found corresponding to that specified by the given thread ID.

EINVAL The value of the *sig* argument is an invalid or unsupported signal number.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **kill(1)**, **pthread_self(3T)**, **pthread_sigmask(3T)**, **raise(3C)**, **attributes(5)**, **standards(5)**

| | |
|----------------------|--|
| NAME | pthread_mutexattr_getprioceiling, pthread_mutexattr_setprioceiling – get and set prioceiling attribute of mutex attribute object |
| SYNOPSIS | <pre>cc - mt [<i>flag</i> ...] <i>file</i> ...- lpthread [<i>library</i> ...] #include <pthread.h> int pthread_mutexattr_setprioceiling(pthread_mutexattr_t * attr, int prioceiling int *oldceiling); int pthread_mutexattr_getprioceiling(const pthread_mutexattr_t * attr, int * prioceiling);</pre> |
| DESCRIPTION | <p>The pthread_mutexattr_getprioceiling() and pthread_mutexattr_setprioceiling() functions, respectively, get and set the priority ceiling attribute of a mutex attribute object pointed to by <i>attr</i> which was previously created by the function pthread_mutexattr_init() .</p> <p>The <i>prioceiling</i> attribute contains the priority ceiling of initialized mutexes. The values of <i>prioceiling</i> will be within the maximum range of priorities defined by SCHED_FIFO.</p> <p>The <i>prioceiling</i> attribute defines the priority ceiling of initialized mutexes, which is the minimum priority level at which the critical section guarded by the mutex is executed. In order to avoid priority inversion, the priority ceiling of the mutex will be set to a priority higher than or equal to the highest priority of all the threads that may lock that mutex. The values of <i>prioceiling</i> will be within the maximum range of priorities defined under the SCHED_FIFO scheduling policy.</p> <p>These functions are not currently supported and will always return ENOSYS .</p> |
| RETURN VALUES | <p>Upon successful completion, the pthread_mutexattr_getprioceiling() and pthread_mutexattr_setprioceiling() functions return 0 . Otherwise, an error number is returned to indicate the error.</p> <p>These functions are not currently supported and will always return ENOSYS .</p> |
| ERRORS | <p>The pthread_mutexattr_getprioceiling() and pthread_mutexattr_setprioceiling() functions will fail if:</p> |

ENOSYS The option `_POSIX_THREAD_PRIO_PROTECT` is not defined and the implementation does not support the function.

The `pthread_mutexattr_getprioceiling()` and `pthread_mutexattr_setprioceiling()` functions may fail if:

EINVAL The value specified by *attr* or *prioceiling* is invalid.

EPERM The caller does not have the privilege to perform the operation.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`pthread_create(3T)`, `pthread_mutex_init(3T)`, `pthread_cond_init(3T)`, `attributes(5)`, `standards(5)`

| | |
|--------------------|---|
| NAME | pthread_mutexattr_getprotocol, pthread_mutexattr_setprotocol – get and set protocol attribute of mutex attribute object |
| SYNOPSIS | <pre> cc - mt [<i>flag</i> ...] <i>file</i> ...- lpthread [<i>library</i> ...] #include <pthread.h> int pthread_mutexattr_setprotocol(pthread_mutexattr_t * attr, int <i>protocol</i>); int pthread_mutexattr_getprotocol(const pthread_mutexattr_t * attr, int * <i>protocol</i>); </pre> |
| DESCRIPTION | <p>The pthread_mutexattr_setprotocol() and pthread_mutexattr_getprotocol() functions, respectively, set and get the protocol attribute of a mutex attribute object pointed to by <i>attr</i> which was previously created by the function pthread_mutexattr_init() .</p> <p>The <i>protocol</i> attribute defines the protocol to be followed in utilizing mutexes. The value of <i>protocol</i> may be one of PTHREAD_PRIO_NONE , PTHREAD_PRIO_INHERIT or PTHREAD_PRIO_PROTECT , which are defined by the header <pthread.h> .</p> <p>When a thread owns a mutex with the PTHREAD_PRIO_NONE protocol attribute, its priority and scheduling are not affected by its mutex ownership.</p> <p>When a thread is blocking higher priority threads because of owning one or more mutexes with the PTHREAD_PRIO_INHERIT protocol attribute, it executes at the higher of its priority or the priority of the highest priority thread waiting on any of the mutexes owned by this thread and initialized with this protocol.</p> <p>When a thread owns one or more mutexes initialized with the PTHREAD_PRIO_PROTECT protocol, it executes at the higher of its priority or the highest of the priority ceilings of all the mutexes owned by this thread and initialized with this attribute, regardless of whether other threads are blocked on any of these mutexes or not.</p> <p>While a thread is holding a mutex which has been initialized with the PRIORITY_INHERIT or PRIORITY_PROTECT protocol attributes, it will not be subject to being moved to the tail of the scheduling queue at its priority in the event that its original priority is changed, such as by a call to sched_setparam() . Likewise, when a thread unlocks a mutex that has been initialized with the</p> |

PRIO_INHERIT or PRIO_PROTECT protocol attributes, it will not be subject to being moved to the tail of the scheduling queue at its priority in the event that its original priority is changed.

If a thread simultaneously owns several mutexes initialized with different protocols, it will execute at the highest of the priorities that it would have obtained by each of these protocols.

When a thread makes a call to **pthread_mutex_lock()**, if the symbol `_POSIX_THREAD_PRIORITY_INHERIT` is defined and the mutex was initialized with the protocol attribute having the value `PTHREAD_PRIORITY_INHERIT`, when the calling thread is blocked because the mutex is owned by another thread, that owner thread will inherit the priority level of the calling thread as long as it continues to own the mutex. The implementation updates its execution priority to the maximum of its assigned priority and all its inherited priorities. Furthermore, if this owner thread itself becomes blocked on another mutex, the same priority inheritance effect will be propagated to this other owner thread, in a recursive manner.

These functions are not currently supported and will always return `ENOSYS`.

RETURN VALUES

Upon successful completion, the **pthread_mutexattr_getprotocol()** and **pthread_mutexattr_setprotocol()** functions return 0. Otherwise, an error number is returned to indicate the error.

These functions are not currently supported and will always return `ENOSYS`.

ERRORS

The **pthread_mutexattr_getprotocol()** and **pthread_mutexattr_setprotocol()** functions will fail if:

- ENOSYS** Neither one of the options `_POSIX_THREAD_PRIORITY_PROTECT` and `_POSIX_THREAD_PRIORITY_INHERIT` is defined and the implementation does not support the function.
- ENOTSUP** The value specified by *protocol* is an unsupported value. The **pthread_mutexattr_getprotocol()** and **pthread_mutexattr_setprotocol()** functions may fail if:
- EINVAL** The value specified by *attr* or *protocol* is invalid.
- EPERM** The caller does not have the privilege to perform the operation.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`pthread_create(3T)`, `pthread_mutex_init(3T)`,
`pthread_cond_init(3T)`, `attributes(5)`, `standards(5)`

| | |
|----------------------|--|
| NAME | pthread_mutexattr_getpshared, pthread_mutexattr_setpshared – get and set process-shared attribute |
| SYNOPSIS | <pre>cc - mt [flag ...] file ...- lpthread [library ...] #include <pthread.h> int pthread_mutexattr_getpshared(const pthread_mutexattr_t * attr, int * pshared); int pthread_mutexattr_setpshared(pthread_mutexattr_t * attr, int pshared);</pre> |
| DESCRIPTION | <p>The pthread_mutexattr_getpshared() function obtains the value of the <i>process-shared</i> attribute from the attributes object referenced by <i>attr</i> . The pthread_mutexattr_setpshared() function is used to set the <i>process-shared</i> attribute in an initialized attributes object referenced by <i>attr</i> .</p> <p>The <i>process-shared</i> attribute is set to <code>PTHREAD_PROCESS_SHARED</code> to permit a mutex to be operated upon by any thread that has access to the memory where the mutex is allocated, even if the mutex is allocated in memory that is shared by multiple processes. If the <i>process-shared</i> attribute is <code>PTHREAD_PROCESS_PRIVATE</code> , the mutex will only be operated upon by threads created within the same process as the thread that initialized the mutex; if threads of differing processes attempt to operate on such a mutex, the behavior is undefined. The default value of the attribute is <code>PTHREAD_PROCESS_PRIVATE</code> .</p> |
| RETURN VALUES | <p>Upon successful completion, pthread_mutexattr_getpshared() returns 0 and stores the value of the <i>process-shared</i> attribute of <i>attr</i> into the object referenced by the <i>pshared</i> parameter. Otherwise, an error number is returned to indicate the error.</p> <p>Upon successful completion, pthread_mutexattr_setpshared() returns 0 . Otherwise, an error number is returned to indicate the error.</p> |
| ERRORS | <p>The pthread_mutexattr_getpshared() and pthread_mutexattr_setpshared() functions may fail if:</p> <p>EINVAL The value specified by <i>attr</i> is invalid.</p> <p>The pthread_mutexattr_setpshared() function will fail if:</p> <p>EINVAL The new value specified for the attribute is outside the range of legal values for that attribute.</p> |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

pthread_create(3T) , **pthread_mutex_init(3T)** ,
pthread_mutexattr_init(3T) , **pthread_cond_init(3T)** ,
attributes(5) , **standards(5)**

| | | | | | | | |
|---------------------------------------|--|-----------------------------------|--|---------------------------------------|--|--------------------------------------|--|
| NAME | pthread_mutexattr_gettype, pthread_mutexattr_settype – get or set a mutex type | | | | | | |
| SYNOPSIS | <pre>cc - mt [flag ...] file ...- lpthread [library ...] #include <pthread.h> int pthread_mutexattr_gettype(pthread_mutexattr_t *attr, int *type); int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);</pre> | | | | | | |
| DESCRIPTION | <p>The pthread_mutexattr_gettype() and pthread_mutexattr_settype() functions respectively get and set the mutex <code>type</code> attribute. This attribute is set in the <code>type</code> parameter to these functions. The default value of the <code>type</code> attribute is <code>PTHREAD_MUTEX_DEFAULT</code>.</p> <p>The type of mutex is contained in the <code>type</code> attribute of the mutex attributes. Valid mutex types include:</p> <table border="0" style="width: 100%;"> <tr> <td style="vertical-align: top; padding-right: 20px;"><code>PTHREAD_MUTEX_NORMAL</code></td> <td>This type of mutex does not detect deadlock. A thread attempting to relock this mutex without first unlocking it will deadlock. Attempting to unlock a mutex locked by a different thread results in undefined behavior. Attempting to unlock an unlocked mutex results in undefined behavior.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 20px;"><code>PTHREAD_MUTEX_ERRORCHECK</code></td> <td>This type of mutex provides error checking. A thread attempting to relock this mutex without first unlocking it will return with an error. A thread attempting to unlock a mutex which another thread has locked will return with an error. A thread attempting to unlock an unlocked mutex will return with an error.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 20px;"><code>PTHREAD_MUTEX_RECURSIVE</code></td> <td>A thread attempting to relock this mutex without first unlocking it will</td> </tr> </table> | <code>PTHREAD_MUTEX_NORMAL</code> | This type of mutex does not detect deadlock. A thread attempting to relock this mutex without first unlocking it will deadlock. Attempting to unlock a mutex locked by a different thread results in undefined behavior. Attempting to unlock an unlocked mutex results in undefined behavior. | <code>PTHREAD_MUTEX_ERRORCHECK</code> | This type of mutex provides error checking. A thread attempting to relock this mutex without first unlocking it will return with an error. A thread attempting to unlock a mutex which another thread has locked will return with an error. A thread attempting to unlock an unlocked mutex will return with an error. | <code>PTHREAD_MUTEX_RECURSIVE</code> | A thread attempting to relock this mutex without first unlocking it will |
| <code>PTHREAD_MUTEX_NORMAL</code> | This type of mutex does not detect deadlock. A thread attempting to relock this mutex without first unlocking it will deadlock. Attempting to unlock a mutex locked by a different thread results in undefined behavior. Attempting to unlock an unlocked mutex results in undefined behavior. | | | | | | |
| <code>PTHREAD_MUTEX_ERRORCHECK</code> | This type of mutex provides error checking. A thread attempting to relock this mutex without first unlocking it will return with an error. A thread attempting to unlock a mutex which another thread has locked will return with an error. A thread attempting to unlock an unlocked mutex will return with an error. | | | | | | |
| <code>PTHREAD_MUTEX_RECURSIVE</code> | A thread attempting to relock this mutex without first unlocking it will | | | | | | |

succeed in locking the mutex. The relocking deadlock which can occur with mutexes of type

`PTHREAD_MUTEX_NORMAL` cannot occur with this type of mutex.

Multiple locks of this mutex require the same number of unlocks to release the mutex before another thread can acquire the mutex. A thread attempting to unlock a mutex which another thread has locked will return with an error. A thread attempting to unlock an unlocked mutex will return with an error. This type of mutex is only supported for mutexes whose process shared attribute is

`PTHREAD_PROCESS_PRIVATE`.

`PTHREAD_MUTEX_DEFAULT`

Attempting to recursively lock a mutex of this type results in undefined behavior. Attempting to unlock a mutex of this type which was not locked by the calling thread results in undefined behavior. Attempting to unlock a mutex of this type which is not locked results in undefined behavior. An implementation is allowed to map this mutex to one of the other mutex types.

RETURN VALUES

If successful, the `pthread_mutexattr_settype()` function returns 0 . Otherwise, an error number is returned to indicate the error.

Upon successful completion, the `pthread_mutexattr_gettype()` function returns 0 and stores the value of the `type` attribute of `attr` into the object referenced by the `type` parameter. Otherwise an error is returned to indicate the error.

ERRORS

The `pthread_mutexattr_gettype()` and `pthread_mutexattr_settype()` functions will fail if:

EINVAL The value `type` is invalid.

EINVAL The value specified by `attr` is invalid.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

pthread_cond_timedwait(3T), **pthread_cond_wait(3T)**,
attributes(5)

NOTES

It is advised that an application should not use a `PTHREAD_MUTEX_RECURSIVE` mutex with condition variables `PTHREAD_MUTEX_RECURSIVE` because the implicit unlock performed for a **pthread_cond_wait()** or **pthread_cond_timedwait()** will not actually release the mutex (if it had been locked multiple times). If this happens, no other thread can satisfy the condition of the predicate.

| | |
|----------------------|---|
| NAME | pthread_mutexattr_init, pthread_mutexattr_destroy – initialize and destroy mutex attributes object |
| SYNOPSIS | <pre>cc - mt [<i>flag</i> ...] <i>file</i> ...- lpthread [<i>library</i> ...] #include <pthread.h> int pthread_mutexattr_init(pthread_mutexattr_t * attr); int pthread_mutexattr_destroy(pthread_mutexattr_t * attr);</pre> |
| DESCRIPTION | <p>The function pthread_mutexattr_init() initializes a mutex attributes object <i>attr</i> with the default value for all of the attributes defined by the implementation.</p> <p>The effect of initializing an already initialized mutex attributes object is undefined.</p> <p>After a mutex attributes object has been used to initialize one or more mutexes, any function affecting the attributes object (including destruction) does not affect any previously initialized mutexes.</p> <p>The pthread_mutexattr_destroy() function destroys a mutex attributes object; the object becomes, in effect, uninitialized. An implementation may cause pthread_mutexattr_destroy() to set the object referenced by <i>attr</i> to an invalid value. A destroyed mutex attributes object can be re-initialized using pthread_mutexattr_init() ; the results of otherwise referencing the object after it has been destroyed are undefined.</p> |
| RETURN VALUES | Upon successful completion, pthread_mutexattr_init() and pthread_mutexattr_destroy() return 0 . Otherwise, an error number is returned to indicate the error. |
| ERRORS | <p>The pthread_mutexattr_init() function may fail if:</p> <p>ENOMEM Insufficient memory exists to initialize the mutex attributes object.</p> <p>The pthread_mutexattr_destroy() function may fail if:</p> <p>EINVAL The value specified by <i>attr</i> is invalid.</p> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`pthread_create(3T)`, `pthread_mutex_init(3T)`,
`pthread_mutexattr_init(3T)`, `pthread_cond_init(3T)`,
`attributes(5)`, `standards(5)`

| | |
|----------------------|--|
| NAME | pthread_mutex_getprioceiling, pthread_mutex_setprioceiling – change the priority ceiling of a mutex |
| SYNOPSIS | <pre>cc - mt [<i>flag</i> ...] <i>file</i> ...- lpthread [<i>library</i> ...] #include <pthread.h> int pthread_mutex_setprioceiling(pthread_mutex_t * <i>mutex</i>, int <i>prioceiling</i>, int * <i>old_ceiling</i>); int pthread_mutex_getprioceiling(const pthread_mutex_t * <i>mutex</i>, int * <i>prioceiling</i>);</pre> |
| DESCRIPTION | <p>The pthread_mutex_getprioceiling() function returns the current priority ceiling of the mutex.</p> <p>The pthread_mutex_setprioceiling() function either locks the mutex if it is unlocked, or blocks until it can successfully lock the mutex, then it changes the mutex's priority ceiling and releases the mutex. When the change is successful, the previous value of the priority ceiling is returned in <i>old_ceiling</i> . The process of locking the mutex need not adhere to the priority protect protocol.</p> <p>If the pthread_mutex_setprioceiling() function fails, the mutex priority ceiling is not changed.</p> <p>These functions are not currently supported and will always return ENOSYS .</p> |
| RETURN VALUES | <p>Upon successful completion, the pthread_mutex_getprioceiling() and pthread_mutex_setprioceiling() functions return 0 . Otherwise, an error number is returned to indicate the error.</p> <p>These functions are not currently supported and will always return ENOSYS .</p> |
| ERRORS | <p>The pthread_mutex_getprioceiling() and pthread_mutex_setprioceiling() functions will fail if:</p> <p>ENOSYS The option <code>_POSIX_THREAD_PRIO_PROTECT</code> is not defined and the implementation does not support the function.</p> <p>The pthread_mutex_getprioceiling() and pthread_mutex_setprioceiling() functions may fail if:</p> <p>EINVAL The priority requested by <i>prioceiling</i> is out of range.</p> |

- EINVAL** The value specified by *mutex* does not refer to a currently existing mutex.
- ENOSYS** The implementation does not support the priority ceiling protocol for mutexes.
- EPERM** The caller does not have the privilege to perform the operation.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`pthread_mutex_init(3T)`, `pthread_mutex_lock(3T)`,
`pthread_mutex_unlock(3T)`, `pthread_mutex_trylock(3T)`,
`attributes(5)`, `standards(5)`

| | |
|----------------------|---|
| NAME | pthread_mutex_init, pthread_mutex_destroy – initialize or destroy a mutex |
| SYNOPSIS | <pre>cc - mt [<i>flag</i> ...] <i>file</i> ...- lpthread [<i>library</i> ...] #include <pthread.h> int pthread_mutex_init(pthread_mutex_t * <i>mutex</i>, const pthread_mutexattr_t * <i>attr</i>); int pthread_mutex_destroy(pthread_mutex_t * <i>mutex</i>); pthread_mutex_t <i>mutex</i> = PTHREAD_MUTEX_INITIALIZER</pre> |
| DESCRIPTION | <p>The pthread_mutex_init() function initializes the mutex referenced by <i>mutex</i> with attributes specified by <i>attr</i>. If <i>attr</i> is <code>NULL</code>, the default mutex attributes are used; the effect is the same as passing the address of a default mutex attributes object. Upon successful initialization, the state of the mutex becomes initialized and unlocked.</p> <p>Attempting to initialize an already initialized mutex results in undefined behavior.</p> <p>The pthread_mutex_destroy() function destroys the mutex object referenced by <i>mutex</i>; the mutex object becomes, in effect, uninitialized. A destroyed mutex object can be re-initialized using pthread_mutex_init(); the results of otherwise referencing the object after it has been destroyed are undefined.</p> <p>It is safe to destroy an initialized mutex that is unlocked. Attempting to destroy a locked mutex results in undefined behavior.</p> <p>In cases where default mutex attributes are appropriate, the macro <code>PTHREAD_MUTEX_INITIALIZER</code> can be used to initialize mutexes that are statically allocated. The effect is equivalent to dynamic initialization by a call to pthread_mutex_init() with parameter <i>attr</i> specified as <code>NULL</code>, except that no error checks are performed.</p> |
| RETURN VALUES | If successful, the pthread_mutex_init() and pthread_mutex_destroy() functions return 0. Otherwise, an error number is returned to indicate the error. |
| ERRORS | <p>The pthread_mutex_init() function may fail if:</p> <p>EINVAL The value specified by <i>attr</i> or <i>mutex</i> is invalid.</p> |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`mutex(3T)`, `pthread_mutex_getprioceiling(3T)`,
`pthread_mutex_lock(3T)`, `pthread_mutex_unlock(3T)`,
`pthread_mutex_setprioceiling(3T)`, `pthread_mutex_trylock(3T)`,
`pthread_mutexattr_getpshared(3T)`,
`pthread_mutexattr_setpshared(3T)` `attributes(5)`, `standards(5)`

| | |
|--------------------|--|
| NAME | pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock – lock or unlock a mutex |
| SYNOPSIS | <pre>cc - mt [<i>flag</i> ...] <i>file</i> ...- lpthread [<i>library</i> ...] #include <pthread.h> int pthread_mutex_lock(pthread_mutex_t * mutex); int pthread_mutex_trylock(pthread_mutex_t * mutex); int pthread_mutex_unlock(pthread_mutex_t * mutex);</pre> |
| DESCRIPTION | <p>The mutex object referenced by mutex is locked by calling pthread_mutex_lock() . If the mutex is already locked, the calling thread blocks until the mutex becomes available. This operation returns with the mutex object referenced by mutex in the locked state with the calling thread as its owner.</p> <p>If the mutex type is <code>PTHREAD_MUTEX_NORMAL</code> , deadlock detection is not provided. Attempting to relock the mutex causes deadlock. If a thread attempts to unlock a mutex that it has not locked or a mutex which is unlocked, undefined behavior results.</p> <p>If the mutex type is <code>PTHREAD_MUTEX_ERRORCHECK</code> , then error checking is provided. If a thread attempts to relock a mutex that it has already locked, an error will be returned. If a thread attempts to unlock a mutex that it has not locked or a mutex which is unlocked, an error will be returned.</p> <p>If the mutex type is <code>PTHREAD_MUTEX_RECURSIVE</code> , then the mutex maintains the concept of a lock count. When a thread successfully acquires a mutex for the first time, the lock count is set to one. Every time a thread relocks this mutex, the lock count is incremented by one. Each time the thread unlocks the mutex, the lock count is decremented by one. When the lock count reaches 0 , the mutex becomes available for other threads to acquire. If a thread attempts to unlock a mutex that it has not locked or a mutex which is unlocked, an error will be returned.</p> <p>If the mutex type is <code>PTHREAD_MUTEX_DEFAULT</code> , attempting to recursively lock the mutex results in undefined behavior. Attempting to unlock the mutex if it was not locked by the calling thread results in undefined behavior.</p> |

Attempting to unlock the mutex if it is not locked results in undefined behavior.

The function **pthread_mutex_trylock()** is identical to `pthread_mutex_lock` except that if the mutex object referenced by `mutex` is currently locked (by any thread, including the current thread), the call returns immediately.

The **pthread_mutex_unlock()** function releases the mutex object referenced by `mutex`. The manner in which a mutex is released is dependent upon the mutex's type attribute. If there are threads blocked on the mutex object referenced by `mutex` when `pthread_mutex_unlock` is called, resulting in the mutex becoming available, the scheduling policy is used to determine which thread shall acquire the mutex. (In the case of `PTHREAD_MUTEX_RECURSIVE` mutexes, the mutex becomes available when the count reaches 0 and the calling thread no longer has any locks on this mutex).

If a signal is delivered to a thread waiting for a mutex, upon return from the signal handler the thread resumes waiting for the mutex as if it was not interrupted.

RETURN VALUES

If successful, the **pthread_mutex_lock()** and **pthread_mutex_unlock()** functions return 0. Otherwise, an error number is returned to indicate the error.

The **pthread_mutex_trylock()** function returns 0 if a lock on the mutex object referenced by `mutex` is acquired. Otherwise, an error number is returned to indicate the error.

ERRORS

The **pthread_mutex_trylock()** function will fail if:

EBUSY The mutex could not be acquired because it was already locked.

The **pthread_mutex_trylock()** and **pthread_mutex_lock()** functions will fail if:

EAGAIN The mutex could not be acquired because the maximum number of recursive locks for `mutex` has been exceeded.

The **pthread_mutex_lock()** function may fail if:

EDEADLK The current thread already owns the mutex.

The **pthread_mutex_unlock()** function may fail if:

EPERM The current thread does not own the mutex.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | `pthread_mutex_init(3T)` , `pthread_mutex_destroy(3T)` ,
`attributes(5)` , `standards(5)`

NOTES | In the current implementation of threads, `pthread_mutex_lock()` ,
`pthread_mutex_unlock()` , `mutex_lock()` , `mutex_unlock()` ,
`pthread_mutex_trylock()` , and `mutex_trylock()` do not validate the mutex
type. Therefore, an uninitialized mutex or a mutex with an invalid type does
not return `EINVAL` . Interfaces for mutexes with an invalid type have
unspecified behavior.

Uninitialized mutexes which are allocated locally may contain junk data. Such
mutexes need to be initialized using `pthread_mutex_init()` or `mutex_init()` .

| NAME | pthread_once – initialize dynamic package | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc - mt [<i>flag...</i>] <i>file...</i>-lpthread [<i>library...</i>] #include <pthread.h> pthread_once_t <i>once_control</i> = PTHREAD_ONCE_INIT; int pthread_once(pthread_once_t *<i>once_control</i>, void (*<i>init_routine</i>, void));</pre> | | | | |
| DESCRIPTION | <p>If any thread in a process with a <i>once_control</i> parameter makes a call to pthread_once(), the first call will summon the init_routine(), but subsequent calls will not. The <i>once_control</i> parameter determines whether the associated initialization routine has been called. The init_routine() is complete upon return of pthread_once().</p> <p>pthread_once() is not a cancellation point; however, if the function init_routine() is a cancellation point and is canceled, the effect on <i>once_control</i> is the same as if pthread_once() had never been called.</p> <p>The constant PTHREAD_ONCE_INIT is defined in the <pthread.h> header.</p> <p>If <i>once_control</i> has automatic storage duration or is not initialized by PTHREAD_ONCE_INIT, the behavior of pthread_once() is undefined.</p> | | | | |
| RETURN VALUES | Upon successful completion, pthread_once() returns 0. Otherwise, an error number is returned to indicate the error. | | | | |
| ERRORS | EINVAL <i>once_control</i> or <i>init_routine</i> is NULL. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | attributes(5) | | | | |
| NOTES | Solaris threads do not offer this functionality. | | | | |

| | |
|----------------------|---|
| NAME | pthread_rwlockattr_getpshared, pthread_rwlockattr_setpshared – get or set process-shared attribute of read-write lock attributes object |
| SYNOPSIS | <pre>cc - mt [flag ...] file ...- lpthread [library ...] #include <pthread.h> int pthread_rwlockattr_getpshared(const pthread_rwlockattr_t *attr, int *pshared); int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr, int pshared);</pre> |
| DESCRIPTION | <p>The <i>process-shared</i> attribute is set to <code>PTHREAD_PROCESS_SHARED</code> to permit a read-write lock to be operated upon by any thread that has access to the memory where the read-write lock is allocated, even if the read-write lock is allocated in memory that is shared by multiple processes. If the <i>process-shared</i> attribute is <code>PTHREAD_PROCESS_PRIVATE</code>, the read-write lock will only be operated upon by threads created within the same process as the thread that initialised the read-write lock; if threads of differing processes attempt to operate on such a read-write lock, the behaviour is undefined. The default value of the <i>process-shared</i> attribute is <code>PTHREAD_PROCESS_PRIVATE</code>.</p> <p>The <code>pthread_rwlockattr_getpshared()</code> function obtains the value of the <i>process-shared</i> attribute from the initialised attributes object referenced by <i>attr</i>. The <code>pthread_rwlockattr_setpshared()</code> function is used to set the <i>process-shared</i> attribute in an initialised attributes object referenced by <i>attr</i>.</p> |
| RETURN VALUES | <p>If successful, the <code>pthread_rwlockattr_setpshared()</code> function returns 0. Otherwise, an error number is returned to indicate the error.</p> <p>Upon successful completion, the <code>pthread_rwlockattr_getpshared()</code> returns 0 and stores the value of the <i>process-shared</i> attribute of <i>attr</i> into the object referenced by the <i>pshared</i> parameter. Otherwise an error number is returned to indicate the error.</p> |
| ERRORS | <p>The <code>pthread_rwlockattr_getpshared()</code> and <code>pthread_rwlockattr_setpshared()</code> functions will fail if:</p> <p>EINVAL The value specified by <i>attr</i> or <i>pshared</i> is invalid.</p> |
| ATTRIBUTES | See <code>attributes(5)</code> for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`pthread_rwlock_init(3T)`, `pthread_rwlock_rdlock(3T)`,
`pthread_rwlock_unlock(3T)`, `pthread_rwlock_wrlock(3T)`,
`pthread_rwlockattr_init(3T)`, `attributes(5)`

| | |
|----------------------|---|
| NAME | pthread_rwlockattr_init, pthread_rwlockattr_destroy – initialize or destroy read-write lock attributes object |
| SYNOPSIS | <pre>cc - mt [<i>flag</i> ...] <i>file</i> ...- lpthread [<i>library</i> ...] #include <pthread.h> int pthread_rwlockattr_init(pthread_rwlockattr_t *attr); int pthread_rwlockattr_destroy(pthread_rwlockattr_t *attr);</pre> |
| DESCRIPTION | <p>The pthread_rwlockattr_init() function initializes a read-write lock attributes object <i>attr</i> with the default value for all of the attributes defined by the implementation.</p> <p>Results are undefined if pthread_rwlockattr_init() is called specifying an already initialized read-write lock attributes object.</p> <p>After a read-write lock attributes object has been used to initialize one or more read-write locks, any function affecting the attributes object (including destruction) does not affect any previously initialized read-write locks.</p> <p>The pthread_rwlockattr_destroy() function destroys a read-write lock attributes object. The effect of subsequent use of the object is undefined until the object is re-initialized by another call to pthread_rwlockattr_init() . An implementation may cause pthread_rwlockattr_destroy() to set the object referenced by <i>attr</i> to an invalid value.</p> |
| RETURN VALUES | If successful, the pthread_rwlockattr_init() and pthread_rwlockattr_destroy() functions return 0 . Otherwise, an error number is returned to indicate the error. |
| ERRORS | <p>The pthread_rwlockattr_init() function will fail if:</p> <p>ENOMEM Insufficient memory exists to initialize the read-write lock attributes object.</p> <p>The pthread_rwlockattr_destroy() function may fail if:</p> <p>EINVAL The value specified by <i>attr</i> is invalid.</p> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`pthread_rwlock_init(3T)`, `pthread_rwlock_rdlock(3T)`,
`pthread_rwlock_unlock(3T)`, `pthread_rwlock_wrlock(3T)`,
`pthread_rwlockattr_getpshared(3T)`, `attributes(5)`

| | |
|--------------------|---|
| NAME | pthread_rwlock_init, pthread_rwlock_destroy – initialize or destroy a read-write lock object |
| SYNOPSIS | <pre>cc - mt [<i>flag</i> ...] <i>file</i> ...- lpthread [<i>library</i> ...] #include <pthread.h> int pthread_rwlock_init(pthread_rwlock_t *rwlock, const pthread_rwlockattr_t *attr); int pthread_rwlock_destroy(pthread_rwlock_t *rwlock); pthread_rwlock_t <i>rwlock</i> =PTHREAD_RWLOCK_INITIALIZER;</pre> |
| DESCRIPTION | <p>The pthread_rwlock_init() function initializes the read-write lock referenced by <i>rwlock</i> with the attributes referenced by <i>attr</i> . If <i>attr</i> is NULL, the default read-write lock attributes are used; the effect is the same as passing the address of a default read-write lock attributes object. Once initialized, the lock can be used any number of times without being re-initialized. Upon successful initialization, the state of the read-write lock becomes initialized and unlocked. Results are undefined if pthread_rwlock_init() is called specifying an already initialized read-write lock. Results are undefined if a read-write lock is used without first being initialized.</p> <p>If the pthread_rwlock_init() function fails, <i>rwlock</i> is not initialized and the contents of <i>rwlock</i> are undefined.</p> <p>The pthread_rwlock_destroy() function destroys the read-write lock object referenced by <i>rwlock</i> and releases any resources used by the lock. The effect of subsequent use of the lock is undefined until the lock is re-initialized by another call to pthread_rwlock_init() . An implementation may cause pthread_rwlock_destroy() to set the object referenced by <i>rwlock</i> to an invalid value. Results are undefined if pthread_rwlock_destroy() is called when any thread holds <i>rwlock</i> . Attempting to destroy an uninitialized read-write lock results in undefined behaviour. A destroyed read-write lock object can be re-initialized using pthread_rwlock_init() ; the results of otherwise referencing the read-write lock object after it has been destroyed are undefined.</p> <p>In cases where default read-write lock attributes are appropriate, the macro PTHREAD_RWLOCK_INITIALIZER can be used to initialize read-write locks that are statically allocated. The effect is equivalent to dynamic initialization by</p> |

a call to **pthread_rwlock_init()** with the parameter *attr* specified as NULL, except that no error checks are performed.

RETURN VALUES

If successful, the **pthread_rwlock_init()** and **pthread_rwlock_destroy()** functions return 0 . Otherwise, an error number is returned to indicate the error.

ERRORS

The **pthread_rwlock_init()** and **pthread_rwlock_init()** functions will fail if:

EINVAL The value specified by *attr* is invalid.

EINVAL The value specified by *rwlock* is invalid.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

pthread_rwlock_rdlock(3T) , **pthread_rwlock_unlock(3T)** ,
pthread_rwlock_wrlock(3T) , **pthread_rwlockattr_init(3T)** ,
attributes(5)

| | |
|--------------------|--|
| NAME | pthread_rwlock_rdlock, pthread_rwlock_tryrdlock – lock or attempt to lock a read-write lock object for reading |
| SYNOPSIS | <pre>cc - mt [<i>flag</i> ...] <i>file</i> ...- lpthread [<i>library</i> ...] #include <pthread.h> int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock); int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);</pre> |
| DESCRIPTION | <p>The pthread_rwlock_rdlock() function applies a read lock to the read-write lock referenced by <i>rwlock</i> . The calling thread acquires the read lock if a writer does not hold the lock and there are no writers blocked on the lock. It is unspecified whether the calling thread acquires the lock when a writer does not hold the lock and there are writers waiting for the lock. If a writer holds the lock, the calling thread will not acquire the read lock. If the read lock is not acquired, the calling thread blocks (that is, it does not return from the pthread_rwlock_rdlock() call) until it can acquire the lock. Results are undefined if the calling thread holds a write lock on <i>rwlock</i> at the time the call is made.</p> <p>Implementations are allowed to favor writers over readers to avoid writer starvation. The current implementation favors writers over readers.</p> <p>A thread may hold multiple concurrent read locks on <i>rwlock</i> (that is, successfully call the pthread_rwlock_rdlock() function <i>n</i> times). If so, the thread must perform matching unlocks (that is, it must call the pthread_rwlock_unlock() function <i>n</i> times).</p> <p>The function pthread_rwlock_tryrdlock() applies a read lock as in the pthread_rwlock_rdlock() function with the exception that the function fails if any thread holds a write lock on <i>rwlock</i> or there are writers blocked on <i>rwlock</i> .</p> <p>Results are undefined if any of these functions are called with an uninitialized read-write lock.</p> <p>If a signal is delivered to a thread waiting for a read-write lock for reading, upon return from the signal handler the thread resumes waiting for the read-write lock for reading as if it was not interrupted.</p> |

RETURN VALUES

If successful, the **pthread_rwlock_rdlock()** function returns 0 . Otherwise, an error number is returned to indicate the error.

The function **pthread_rwlock_tryrdlock()** returns 0 if the lock for reading on the read-write lock object referenced by *rwlock* is acquired. Otherwise an error number is returned to indicate the error.

ERRORS

The **pthread_rwlock_tryrdlock()** function will fail if:

EBUSY The read-write lock could not be acquired for reading because a writer holds the lock or was blocked on it.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

pthread_rwlock_init(3T) , **pthread_rwlock_wrlock(3T)** ,
pthread_rwlockattr_init(3T) , **pthread_rwlock_unlock(3T)** ,
attributes(5)

| NAME | pthread_rwlock_unlock – unlock a read-write lock object | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc - mt [<i>flag...</i>] <i>file...</i>- lpthread [<i>library...</i>] #include <pthread.h> int pthread_rwlock_unlock(pthread_rwlock_t *<i>rwlock</i>);</pre> | | | | |
| DESCRIPTION | <p>The pthread_rwlock_unlock() function is called to release a lock held on the read-write lock object referenced by <i>rwlock</i>. Results are undefined if the read-write lock <i>rwlock</i> is not held by the calling thread.</p> <p>If this function is called to release a read lock from the read-write lock object and there are other read locks currently held on this read-write lock object, the read-write lock object remains in the read locked state. If this function releases the calling thread's last read lock on this read-write lock object, then the calling thread is no longer one of the owners of the object. If this function releases the last read lock for this read-write lock object, the read-write lock object will be put in the unlocked state with no owners.</p> <p>If this function is called to release a write lock for this read-write lock object, the read-write lock object will be put in the unlocked state with no owners.</p> <p>If the call to the pthread_rwlock_unlock() function results in the read-write lock object becoming unlocked and there are multiple threads waiting to acquire the read-write lock object for writing, the scheduling policy is used to determine which thread acquires the read-write lock object for writing. If there are multiple threads waiting to acquire the read-write lock object for reading, the scheduling policy is used to determine the order in which the waiting threads acquire the read-write lock object for reading. If there are multiple threads blocked on <i>rwlock</i> for both read locks and write locks, it is unspecified whether the readers acquire the lock first or whether a writer acquires the lock first.</p> <p>Results are undefined if any of these functions are called with an uninitialized read-write lock.</p> | | | | |
| RETURN VALUES | If successful, the pthread_rwlock_unlock() function returns 0. Otherwise, an error number is returned to indicate the error. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |

SEE ALSO

pthread_rwlock_init(3T), pthread_rwlock_rdlock(3T),
pthread_rwlock_wrlock(3T), pthread_rwlockattr_init(3T),
attributes(5)

| | |
|----------------------|--|
| NAME | pthread_rwlock_wrlock, pthread_rwlock_trywrlock – lock or attempt to lock a read-write lock object for writing |
| SYNOPSIS | <pre>cc - mt [<i>flag</i> ...] <i>file</i> ...- lpthread [<i>library</i> ...] #include <pthread.h> int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock); int pthread_rwlock_trywrlock(pthread_rwlock_t *rwlock);</pre> |
| DESCRIPTION | <p>The pthread_rwlock_wrlock() function applies a write lock to the read-write lock referenced by <i>rwlock</i> . The calling thread acquires the write lock if no other thread (reader or writer) holds the read-write lock <i>rwlock</i> . Otherwise, the thread blocks (that is, does not return from the pthread_rwlock_wrlock() call) until it can acquire the lock. Results are undefined if the calling thread holds the read-write lock (whether a read or write lock) at the time the call is made.</p> <p>Implementations are allowed to favor writers over readers to avoid writer starvation. The current implementation favors writers over readers.</p> <p>The function pthread_rwlock_trywrlock() applies a write lock like the pthread_rwlock_wrlock() function, with the exception that the function fails if any thread currently holds <i>rwlock</i> (for reading or writing).</p> <p>Results are undefined if any of these functions are called with an uninitialized read-write lock.</p> <p>If a signal is delivered to a thread waiting for a read-write lock for writing, upon return from the signal handler the thread resumes waiting for the read-write lock for writing as if it was not interrupted.</p> |
| RETURN VALUES | <p>If successful, the pthread_rwlock_wrlock() function returns 0 . Otherwise, an error number is returned to indicate the error.</p> <p>The function pthread_rwlock_trywrlock() returns 0 if the lock for writing on the read-write lock object referenced by <i>rwlock</i> is acquired. Otherwise an error number is returned to indicate the error.</p> |
| ERRORS | <p>The pthread_rwlock_trywrlock() function will fail if:</p> <p>EBUSY The read-write lock could not be acquired for writing because it was already locked for reading or writing.</p> |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`pthread_rwlock_init(3T)`, `pthread_rwlock_unlock(3T)`,
`pthread_rwlockattr_init(3T)`, `pthread_rwlock_rdlock(3T)`,
`attributes(5)`

NAME pthread_self – get calling thread's ID

SYNOPSIS cc - mt [*flag...*] *file...*- lpthread [*library...*]

```
#include <pthread.h>
pthread_t pthread_self(void);
```

DESCRIPTION The **pthread_self()** function returns the thread ID of the calling thread.

ERRORS No errors are defined.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **pthread_create(3T)**, **pthread_equal(3T)**, **attributes(5)**, **standards(5)**

| | |
|----------------------|--|
| NAME | pthread_setcancelstate – enable or disable cancellation |
| SYNOPSIS | <pre>cc - mt [flag...] file...- lpthread [library...] #include <pthread.h> int pthread_setcancelstate(intstate, int *oldstate);</pre> |
| DESCRIPTION | <p>pthread_setcancelstate() atomically sets the calling thread's cancellation state to the specified <i>state</i> and, if <i>oldstate</i> is not <code>NULL</code>, stores the previous cancellation <i>state</i> in <i>oldstate</i>.</p> <p>The <i>state</i> can be either of the following:</p> <p><code>PTHREAD_CANCEL_ENABLE</code> This is the default. When cancellation is deferred (deferred cancellation is also the default), cancellation occurs when the target thread reaches a cancellation point and a cancel is pending. When cancellation is asynchronous, receipt of a <code>pthread_cancel(3T)</code> call causes immediate cancellation.</p> <p><code>PTHREAD_CANCEL_DISABLE</code></p> <p>When cancellation is deferred, all cancellation requests to the target thread are held pending. When cancellation is asynchronous, all cancellation requests to the target thread are held pending; as soon as cancellation is re-enabled, pending cancellations are executed immediately.</p> <p>See <code>cancellation(3T)</code> for the definition of a cancellation point and a discussion of cancellation concepts. See <code>pthread_setcanceltype(3T)</code> for explanations of deferred and asynchronous cancellation.</p> <p>pthread_setcancelstate() is a cancellation point when it is called with <code>PTHREAD_CANCEL_ENABLE</code> and the cancellation type is <code>PTHREAD_CANCEL_ASYNCHRONOUS</code>.</p> |
| RETURN VALUES | Upon successful completion, <code>pthread_setcancelstate()</code> , returns 0. Otherwise, an error number is returned to indicate the error. |
| ERRORS | The <code>pthread_setcancelstate()</code> function will fail if: |

EINVAL The specified *state* is not `PTHREAD_CANCEL_ENABLE` or `PTHREAD_CANCEL_DISABLE`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`cancellation(3T)`, `condition(3T)`, `pthread_cancel(3T)`, `pthread_cleanup_pop(3T)`, `pthread_cleanup_push(3T)`, `pthread_exit(3T)`, `pthread_join(3T)`, `pthread_setcanceltype(3T)`, `pthread_testcancel(3T)`, `setjmp(3C)`, `attributes(5)`

| | | | | | |
|----------------------------|---|-------------------------|--|----------------------------|---|
| NAME | pthread_setcanceltype – set the cancellation type of a thread | | | | |
| SYNOPSIS | <pre>cc - mt [<i>flag...</i>] <i>file...</i> -lpthread [<i>library...</i>] #include <pthread.h> int pthread_setcanceltype(inttype, int *oldtype);</pre> | | | | |
| DESCRIPTION | <p>pthread_setcanceltype() atomically sets the calling thread's cancellation type to the specified <code>type</code> and, if <code>oldtype</code> is not <code>NULL</code>, stores the previous cancellation type in <code>oldtype</code>. The <code>type</code> can be either of the following:</p> <table border="0"> <tr> <td style="vertical-align: top; padding-right: 20px;">PTHREAD_CANCEL_DEFERRED</td> <td>This is the default. When cancellation is enabled (enabled cancellation is also the default), cancellation occurs when the target thread reaches a cancellation point and a cancel is pending. When cancellation is disabled, all cancellation requests to the target thread are held pending.</td> </tr> <tr> <td style="vertical-align: top; padding-right: 20px;">PTHREAD_CANCEL_ASYNCRONOUS</td> <td>When cancellation is enabled, receipt of a pthread_cancel(3T) call causes immediate cancellation. When cancellation is disabled, all cancellation requests to the target thread are held pending; as soon as cancellation is re-enabled, pending cancellations are executed immediately.</td> </tr> </table> <p>See cancellation(3T) for the definition of a cancellation point and a discussion of cancellation concepts. See pthread_setcancelstate(3T) for explanations of enabling and disabling cancellation.</p> <p>pthread_setcanceltype() is a cancellation point if <code>type</code> is called with <code>PTHREAD_CANCEL_ASYNCRONOUS</code> and the cancellation state is <code>PTHREAD_CANCEL_ENABLE</code>.</p> | PTHREAD_CANCEL_DEFERRED | This is the default. When cancellation is enabled (enabled cancellation is also the default), cancellation occurs when the target thread reaches a cancellation point and a cancel is pending. When cancellation is disabled, all cancellation requests to the target thread are held pending. | PTHREAD_CANCEL_ASYNCRONOUS | When cancellation is enabled, receipt of a pthread_cancel(3T) call causes immediate cancellation. When cancellation is disabled, all cancellation requests to the target thread are held pending; as soon as cancellation is re-enabled, pending cancellations are executed immediately. |
| PTHREAD_CANCEL_DEFERRED | This is the default. When cancellation is enabled (enabled cancellation is also the default), cancellation occurs when the target thread reaches a cancellation point and a cancel is pending. When cancellation is disabled, all cancellation requests to the target thread are held pending. | | | | |
| PTHREAD_CANCEL_ASYNCRONOUS | When cancellation is enabled, receipt of a pthread_cancel(3T) call causes immediate cancellation. When cancellation is disabled, all cancellation requests to the target thread are held pending; as soon as cancellation is re-enabled, pending cancellations are executed immediately. | | | | |
| RETURN VALUES | Upon successful completion, the pthread_setcanceltype() function returns 0. Otherwise, an error number is returned to indicate the error. | | | | |
| ERRORS | <p>The pthread_setcanceltype() function will fail if:</p> <table border="0"> <tr> <td style="vertical-align: top; padding-right: 20px;">EINVAL</td> <td>The specified <code>type</code> is not <code>PTHREAD_CANCEL_DEFERRED</code> or <code>PTHREAD_CANCEL_ASYNCRONOUS</code>.</td> </tr> </table> | EINVAL | The specified <code>type</code> is not <code>PTHREAD_CANCEL_DEFERRED</code> or <code>PTHREAD_CANCEL_ASYNCRONOUS</code> . | | |
| EINVAL | The specified <code>type</code> is not <code>PTHREAD_CANCEL_DEFERRED</code> or <code>PTHREAD_CANCEL_ASYNCRONOUS</code> . | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`cancellation(3T)`, `condition(3T)`, `pthread_cancel(3T)`,
`pthread_cleanup_pop(3T)`, `pthread_cleanup_push(3T)`,
`pthread_exit(3T)`, `pthread_join(3T)`, `pthread_setcancelstate(3T)`,
`pthread_testcancel(3T)`, `setjmp(3C)`, `attributes(5)`

| | |
|----------------------|--|
| NAME | pthread_sigmask - change or examine calling thread's signal mask |
| SYNOPSIS | <pre>cc - mt [<i>flag...</i>] <i>file...</i>- lpthread [<i>library...</i>] #include <pthread.h> #include <signal.h> int pthread_sigmask(int <i>how</i>, const sigset_t *<i>set</i>, sigset_t *<i>oset</i>);</pre> |
| DESCRIPTION | <p>The pthread_sigmask() function changes or examines a calling thread's signal mask. Each thread has its own signal mask. A new thread inherits the calling thread's signal mask and priority; however, pending signals are not inherited. Signals pending for a new thread will be empty.</p> <p>If the value of the argument <i>set</i> is not NULL, <i>set</i> points to a set of signals that can modify the currently blocked set. If the value of <i>set</i> is NULL, the value of <i>how</i> is insignificant and the thread's signal mask is unmodified; thus, pthread_sigmask() can be used to inquire about the currently blocked signals.</p> <p>The value of the argument <i>how</i> specifies the method in which the set is changed and takes one of the following values:</p> <p>SIG_BLOCK <i>set</i> corresponds to a set of signals to block. They are added to the current signal mask.</p> <p>SIG_UNBLOCK <i>set</i> corresponds to a set of signals to unblock. These signals are deleted from the current signal mask.</p> <p>SIG_SETMASK <i>set</i> corresponds to the new signal mask. The current signal mask is replaced by <i>set</i>.</p> <p>If the value of <i>oset</i> is not NULL, it points to the location where the previous signal mask is stored.</p> |
| RETURN VALUES | Upon successful completion, the pthread_sigmask() function returns 0. Otherwise, it returns a non-zero value. |
| ERRORS | <p>The pthread_sigmask() function will fail if:</p> <p>EINVAL The value of <i>how</i> is not defined and <i>oset</i> is NULL.</p> |
| EXAMPLES | <p>EXAMPLE 1 The following example shows how to create a default thread that can serve as a signal catcher/handler with its own signal mask. <i>new</i> will have a different value from the creator's signal mask.</p> <p>As POSIX threads and Solaris threads are fully compatible even within the same process, this example uses pthread_create(3T) if you execute <code>a.out 0</code>, or thr_create(3T) if you execute <code>a.out 1</code>.</p> <p>In this example:</p> |

- **sigemptyset(3C)** initializes a null signal set, new. **sigaddset(3C)** packs the signal, **SIGINT**, into that new set.
- Either **pthread_sigmask()** or **thr_sigsetmask()** is used to mask the signal, **SIGINT** (CTRL-C), from the calling thread, which is **main()**. The signal is masked to guarantee that only the new thread will receive this signal.
- **pthread_create()** or **thr_create()** creates the signal-handling thread.
- Using **pthread_join(3T)** or **thr_join(3T)**, **main()** then waits for the termination of that signal-handling thread, whose ID number is **user_threadID**; after which, **main()** will **sleep(3C)** for 2 seconds, and then the program terminates.
- The signal-handling thread, handler:
 - Assigns the handler **interrupt()** to handle the signal **SIGINT**, by the call to **sigaction(2)**.
 - Resets its own signal set to *not block* the signal, **SIGINT**.
 - Sleeps for 8 seconds to allow time for the user to deliver the signal, **SIGINT**, by pressing the CTRL-C.

```

/* cc thisfile.c -lthread -lpthread */
#define _REENTRANT /* basic first 3-lines for threads */
#include <pthread.h>
#include <thread.h>
thread_t user_threadID;
sigset_t new;
void *handler(), interrupt();

main( int argc, char *argv[] ) {
    test_argv(argv[1]);

    sigemptyset(&new);
    sigaddset(&new, SIGINT);
    switch(*argv[1]) {

        case '0': /* POSIX */
            pthread_sigmask(SIG_BLOCK, &new, NULL);
            pthread_create(&user_threadID, NULL, handler, argv[1]);
            pthread_join(user_threadID, NULL);
            break;

        case '1': /* Solaris */
            thr_sigsetmask(SIG_BLOCK, &new, NULL);
            thr_create(NULL, 0, handler, argv[1], 0, &user_threadID);
            thr_join(user_threadID, NULL, NULL);
            break;
    } /* switch */

    printf("thread handler, # %d, has exited\n",user_threadID);
    sleep(2);
    printf("main thread, # %d is done\n", thr_self());
} /* end main */

```

```

struct sigaction act;

void *
handler(char argv[])
{
    act.sa_handler = interrupt;
    sigaction(SIGINT, &act, NULL);
    switch(*argv) {
        case '0': /* POSIX */
            pthread_sigmask(SIG_UNBLOCK, &new, NULL);
            break;
        case '1': /* Solaris */
            thr_sigsetmask(SIG_UNBLOCK, &new, NULL);
            break;
    }
    printf("\n Press CTRL-C to deliver SIGINT signal to the process\n");
    sleep(8); /* give user time to hit CTRL-C */
}

void
interrupt(int sig)
{
    printf("thread %d caught signal %d\n", thr_self(), sig);
}

void test_argv(char argv[]) {
    if(argv == NULL) {
        printf("use 0 as arg1 to use thr_create();\n \
or use 1 as arg1 to use pthread_create()\n");
        exit(NULL);
    }
}

```

EXAMPLE 2

In the last example, the `handler` thread served as a signal-handler while also taking care of activity of its own (in this case, sleeping, although it could have been some other activity). A thread could be completely dedicated to signal-handling simply by waiting for the delivery of a selected signal by blocking with `sigwait(2)`. The two subroutines in the previous example, `handler()` and `interrupt()`, could have been replaced with the following routine:

```

void *
handler()
{ int signal;
  printf("thread %d is waiting for you to press the CTRL-C keys\n", thr_self());
  sigwait(&new, &signal);
  printf("thread %d has received the signal %d \n", thr_self(), signal);
}
/* pthread_create() and thr_create() would use NULL instead of argv[1]
   for the arg passed to handler() */

```

In this routine, one thread is dedicated to catching and handling the signal specified by the set `new`, which allows `main()` and all of its other sub-threads,

created *after* `pthread_sigmask()` or `thr_sigsetmask()` masked that signal, to continue uninterrupted. Any use of `sigwait(2)` should be such that all threads block the signals passed to `sigwait(2)` at all times. Only the thread that calls `sigwait()` will get the signals. The call to `sigwait(2)` takes two arguments.

For this type of background dedicated signal-handling routine, you may wish to use a Solaris daemon thread by passing the argument, `THR_DAEMON`, to `thr_create(3T)`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------------|
| MT-Level | MT-Safe and Async-Signal-Safe |

SEE ALSO

`sigaction(2)`, `sigprocmask(2)`, `sigwait(2)`, `cond_wait(3T)`, `pthread_create(3T)`, `pthread_join(3T)`, `pthread_self(3T)`, `sigsetops(3C)`, `sleep(3C)`, `attributes(5)`, `standards(5)`

NOTES

It is not possible to block signals that cannot be ignored (see `sigaction(2)`). If using the threads library, it is not possible to block the signals `SIGLWP` or `SIGCANCEL`, which are reserved by the threads library. Additionally, it is impossible to unblock the signal `SIGWAITING`, which is always blocked on all threads. This restriction is quietly enforced by the threads library.

Using `sigwait(2)` in a dedicated thread allows asynchronously generated signals to be managed synchronously; however, `sigwait(2)` should never be used to manage synchronously generated signals.

Synchronously generated signals are exceptions that are generated by a thread and are directed at the thread causing the exception. Since `sigwait()` blocks waiting for signals, the blocking thread cannot receive a synchronously generated signal.

If `sigprocmask(2)` is used in a multi-threaded program, it will be the same as if `pthread_sigmask()` has been called. POSIX leaves the semantics of the call to `sigprocmask(2)` unspecified in a multi-threaded process, so programs that care about POSIX portability should not depend on this semantic.

If a signal is delivered while a thread is waiting on a condition variable, the `cond_wait()` will be interrupted (see `cond_wait(3T)`) and the handler will be executed. The handler should assume that the lock protecting the condition variable is held.

Although **pthread_sigmask()** is Async-Signal-Safe with respect to the Solaris environment, this safeness is not guaranteed to be portable to other POSIX domains.

Signals which are generated synchronously should not be masked. If such a signal is blocked and delivered, the receiving process is killed.

A thread directed `SIGALRM` generated because of a realtime interval timer or process alarm clock is not maskable by a signal masking function, such as `thr_sigsetmask(3T)`, or `sigprocmask(2)`. See `alarm(2)` and `setitimer(2)`.

| NAME | pthread_testcancel – create cancellation point in the calling thread | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc - mt [flag...] file...- lpthread [library...] #include <pthread.h> void pthread_testcancel ();</pre> | | | | |
| DESCRIPTION | <p>The pthread_testcancel() function forces testing for cancellation. This is useful when you need to execute code that runs for long periods without encountering cancellation points; such as a library routine that executes long-running computations without cancellation points. This type of code can block cancellation for unacceptable long periods of time. One strategy for avoiding blocking cancellation for long periods, is to insert calls to pthread_testcancel() in the long-running computation code and to setup a cancellation handler in the library code, if required.</p> | | | | |
| RETURN VALUES | The pthread_testcancel() function returns a void. | | | | |
| ERRORS | The pthread_testcancel() function does not return errors. | | | | |
| EXAMPLES | EXAMPLE 1 See cancellation(3T) for an example of using pthread_testcancel() to force testing for cancellation and a discussion of cancellation concepts. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | Intro(3), cancellation(3T) , condition(3T) , pthread_cleanup_pop(3T) , pthread_cleanup_push(3T) , pthread_exit(3T) , pthread_join(3T) , pthread_setcancelstate(3T) , pthread_setcanceltype(3T) , setjmp(3C) , attributes(5) | | | | |
| NOTES | <p>pthread_testcancel() has no effect if cancellation is disabled.</p> <p>Use pthread_testcancel() with pthread_setcanceltype() called with its canceltype set to PTHREAD_CANCEL_DEFERRED. pthread_testcancel() operation is undefined if pthread_setcanceltype() was called with its canceltype argument set to PTHREAD_CANCEL_ASYNCRONOUS.</p> <p>It is possible to kill a thread when it is holding a resource, such as lock or allocated memory. If that thread has not setup a cancellation cleanup handler</p> | | | | |

to release the held resource, the application is "cancel-unsafe". See **attributes(5)** for a discussion of Cancel-Safety, Deferred-Cancel-Safety, and Asynchronous-Cancel-Safety.

| NAME | ptsname – get name of the slave pseudo-terminal device | | | | |
|----------------------|---|----------------|-----------------|----------|------|
| SYNOPSIS | #include <stdlib.h> char *ptsname(int <i>fildev</i>); | | | | |
| DESCRIPTION | The ptsname() function returns the name of the slave pseudo-terminal device associated with a master pseudo-terminal device. <i>fildev</i> is a file descriptor returned from a successful open of the master device. ptsname() returns a pointer to a string containing the null-terminated path name of the slave device of the form <i>/dev/pts/N</i> , where <i>N</i> is a non-negative integer. | | | | |
| RETURN VALUES | Upon successful completion, the function ptsname() returns a pointer to a string which is the name of the pseudo-terminal slave device. This value points to a static data area that is overwritten by each call to ptsname() . Upon failure, ptsname() returns <code>NULL</code> . This could occur if <i>fildev</i> is an invalid file descriptor or if the slave device name does not exist in the file system. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe | | | | |
| SEE ALSO | open(2) , grantpt(3C) , ttynname(3C) , unlockpt(3C) , attributes(5) <i>STREAMS Programming Guide</i> | | | | |

| NAME | putenv – change or add value to environment | | | | |
|----------------------|--|----------------|-----------------|----------|------|
| SYNOPSIS | <pre>#include <stdlib.h> int putenv(char *string);</pre> | | | | |
| DESCRIPTION | <p>The putenv() function makes the value of the environment variable <i>name</i> equal to <i>value</i> by altering an existing variable or creating a new one. In either case, the string pointed to by <i>string</i> becomes part of the environment, so altering the string will change the environment.</p> <p>The <i>string</i> argument points to a string of the form <i>name=value</i>. The space used by <i>string</i> is no longer used once a new string-defining <i>name</i> is passed to putenv().</p> <p>The putenv() function uses malloc(3C) to enlarge the environment.</p> <p>After putenv() is called, environment variables are not in alphabetical order.</p> | | | | |
| RETURN VALUES | The putenv() functions returns a non-zero value if it was unable to obtain enough space using malloc(3C) for an expanded environment. Otherwise, 0 is returned. | | | | |
| ERRORS | <p>The putenv() function may fail if:</p> <p>ENOMEM Insufficient memory was available.</p> | | | | |
| USAGE | <p>The putenv() function can be safely called from multithreaded programs. Caution must be exercised when using this function and getenv(3C) in multithreaded programs. These functions examine and modify the environment list, which is shared by all threads in a program. The system prevents the list from being accessed simultaneously by two different threads. It does not, however, prevent two threads from successively accessing the environment list using putenv() or getenv().</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe | | | | |
| SEE ALSO | exec(2) , getenv(3C) , malloc(3C) , attributes(5) , environ(5) | | | | |
| WARNINGS | The <i>string</i> argument should not be an automatic variable. It should be declared static if it is declared within a function because it cannot be automatically declared. A potential error is to call putenv() with a pointer to an automatic | | | | |

variable as the argument and to then exit the calling function while *string* is still part of the environment.

| | |
|----------------------|---|
| NAME | putp, tputs – apply padding information and output string |
| SYNOPSIS | <pre>#include <curses.h> int putp(const char * str); int tputs(const char * str, int affcnt, int (* putfunc)(int));</pre> |
| PARAMETERS | <p>str Is a pointer to a <code>terminfo</code> variable or return value from <code>tgetstr(3XC)</code>, <code>tgoto(3XC)</code>, <code>tigetstr(3XC)</code>, or <code>tparm(3XC)</code>.</p> <p>affcnt Is the number of lines affected, or 1 if not relevant.</p> <p>putfunc Is the output function.</p> |
| DESCRIPTION | <p>The <code>putp()</code> and <code>tputs()</code> functions are low-level functions used to deal directly with the <code>terminfo</code> database. The use of appropriate X/Open Curses functions is recommended for most situations.</p> <p>The <code>tputs()</code> function adds padding information and then outputs <code>str</code>. <code>str</code> must be a <code>terminfo</code> string variable or the result value from <code>tgetstr()</code>, <code>tgoto()</code>, <code>tigetstr()</code>, or <code>tparm()</code>. The <code>tputs()</code> function replaces the padding specification (if one exists) with enough characters to produce the specified delay. Characters are output one at a time to <code>putfunc</code>, a user-specified function similar to <code>putchar(3S)</code>.</p> <p>The <code>putp()</code> function calls <code>tputs()</code> as follows:</p> <pre>tputs(str , 1, putchar)</pre> |
| RETURN VALUES | On success, these functions return <code>OK</code> . |
| ERRORS | None. |
| USAGE | The output of <code>putp()</code> goes to <code>stdout</code> , not to the file descriptor, <code>fildev</code> , specified in <code>setupterm(3XC)</code> . |
| SEE ALSO | <code>putchar(3S)</code> , <code>setupterm(3XC)</code> , <code>tgetent(3XC)</code> , <code>tigetflag(3XC)</code> , <code>terminfo(4)</code> |

| NAME | putpwent – write password file entry | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | #include <pwd.h> int putpwent (const struct passwd *p, FILE *f); | | | | |
| DESCRIPTION | The putpwent() function is the inverse of getpwent() . See getpwnam(3C) . Given a pointer to a <code>passwd</code> structure created by getpwent() , getpwuid() , or getpwnam() , putpwent() writes a line on the stream <i>f</i> that matches the format of <code>/etc/passwd</code> . | | | | |
| RETURN VALUES | The putpwent() function returns a non-zero value if an error was detected during its operation. Otherwise, it returns 0. | | | | |
| USAGE | The putpwent() function is of limited utility, since most password files are maintained as Network Information Service (NIS) files that cannot be updated with this function. For this reason, the use of this function is discouraged. If used at all, it should be used with putspent(3C) to update the shadow file. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | getpwnam(3C) , putspent(3C) , attributes(5) | | | | |

| NAME | puts, fputs – put a string on a stream | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <stdio.h> int puts(const char * s); int fputs(const char * s, FILE * stream);</pre> | | | | |
| DESCRIPTION | <p>The puts() function writes the string pointed to by <i>s</i>, followed by a <code>NEWLINE</code> character, to the standard output stream <code>stdout</code> (see <code>intro(3)</code>).</p> <p>The fputs() function writes the null-terminated string pointed to by <i>s</i> to the named output <i>stream</i>.</p> <p>Neither function writes the terminating null character.</p> | | | | |
| RETURN VALUES | On success both routines return the number of characters written; otherwise they return <code>EOF</code> . | | | | |
| ERRORS | <p>The puts() and fputs() functions will fail if either the <i>stream</i> is unbuffered or the <i>stream</i>'s buffer needed to be flushed and:</p> <p>EFBIG The file is a regular file and an attempt was made to write at or beyond the offset maximum.</p> | | | | |
| ATTRIBUTES | See <code>attributes(5)</code> for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | <code>write(2)</code> , <code>intro(3)</code> , <code>fclose(3S)</code> , <code>ferror(3S)</code> , <code>fopen(3S)</code> , <code>printf(3S)</code> , <code>putc(3S)</code> , <code>stdio(3S)</code> , <code>attributes(5)</code> | | | | |

| NAME | putspent – write shadow password file entry | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | <pre>#include <shadow.h> int putspent(const struct spwd *p, FILE *fp);</pre> | | | | |
| DESCRIPTION | <p>The putspent() function is the inverse of getspent(). See getspnam(3C). Given a pointer to a <code>spwd</code> structure created by getspent() or getspnam(), putspent() writes a line on the stream <code>fp</code> that matches the format of <code>/etc/shadow</code>.</p> <p>The <code>spwd</code> structure contains the following members:</p> <pre>char *sp_namp; char *sp_pwdp; long sp_lstchg; long sp_min; long sp_max; long sp_warn; long sp_inact; long sp_expire; unsigned long sp_flag;</pre> <p>If the <code>sp_min</code>, <code>sp_max</code>, <code>sp_lstchg</code>, <code>sp_warn</code>, <code>sp_inact</code>, or <code>sp_expire</code> member of the <code>spwd</code> structure is <code>-1</code>, or if <code>sp_flag</code> is <code>0</code>, the corresponding <code>/etc/shadow</code> field is cleared.</p> | | | | |
| RETURN VALUES | The putspent() function returns a non-zero value if an error was detected during its operation. Otherwise, it returns <code>0</code> . | | | | |
| USAGE | Since this function is for internal use only, compatibility is not guaranteed. For this reason, its use is discouraged. If used at all, it should be used with putpwent(3C) to update the password file. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | getpwnam(3C) , getspnam(3C) , putpwent(3C) , attributes(5) | | | | |

NAME | putws – convert a string of Process Code characters to EUC characters

SYNOPSIS | #include <stdio.h>
#include <wdec.h>

| int putws(wchar_t *s);

DESCRIPTION | The **putws()** function converts the Process Code string (terminated by a (wchar_t) NULL) pointed to by *s*, to an Extended Unix Code (EUC) string followed by a NEWLINE character, and writes it to the standard output stream `stdout`. It does not write the terminal null character.

RETURN VALUES | The **putws()** function returns the number of Process Code characters transformed and written. It returns EOF if it attempts to write to a file that has not been opened for writing.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **ferror(3S)**, **fopen(3S)**, **fread(3S)**, **getws(3S)**, **printf(3S)**, **putwc(3S)**, **attributes(5)**

| | |
|--------------------|---|
| NAME | qsort – quick sort |
| SYNOPSIS | <pre>#include <stdlib.h> void qsort(void *base, size_t nel, size_t width, int (*compar)(const void *, const void *));</pre> |
| DESCRIPTION | <p>The qsort() function is an implementation of the quick-sort algorithm. It sorts a table of data in place. The contents of the table are sorted in ascending order according to the user-supplied comparison function.</p> <p>The <i>base</i> argument points to the element at the base of the table. The <i>nel</i> argument is the number of elements in the table. The <i>width</i> argument specifies the size of each element in bytes. The <i>compar</i> argument is the name of the comparison function, which is called with two arguments that point to the elements being compared.</p> <p>The function must return an integer less than, equal to, or greater than zero to indicate if the first argument is to be considered less than, equal to, or greater than the second argument.</p> <p>The contents of the table are sorted in ascending order according to the user supplied comparison function.</p> |
| EXAMPLES | <p>EXAMPLE 1 Program sorts.</p> <p>The following program sorts a simple array:</p> <pre>static int intcompare(int *i, int *j) { if (*i > *j) return (1); if (*i < *j) return (-1); return (0); } main() { int a[10]; int i; a[0] = 9; a[1] = 8; a[2] = 7; a[3] = 6; a[4] = 5; a[5] = 4; a[6] = 3; a[7] = 2; a[8] = 1; a[9] = 0; qsort((char *) a, 10, sizeof(int), intcompare);</pre> |

```

        for (i=0; i<10; i++) printf(" %d",a[i]);
        printf("\n");
    }

```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

sort(1), **bsearch(3C)**, **lsearch(3C)**, **string(3C)**, **attributes(5)**

NOTES

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

The relative order in the output of two items that compare as equal is unpredictable.

| NAME | raise – send signal to program | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <code>#include <signal.h></code> <code>int raise(int sig);</code> | | | | |
| DESCRIPTION | The raise() function sends the signal <i>sig</i> to the executing program. It uses the kill() function to send the signal to the executing program, as follows: <code>kill(getpid(), sig);</code> See the kill(2) manual page for a detailed list of failure conditions and the signal(3C) manual page for a list of signals. | | | | |
| RETURN VALUES | Upon successful completion, 0 is returned. Otherwise, -1 is returned and errno is set to indicate the error. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | getpid(2) , kill(2) , signal(3C) , attributes(5) | | | | |

| | |
|--------------------|--|
| NAME | rand, srand – simple random number generator |
| SYNOPSIS | <pre>/usr/ucb/cc [<i>flag</i> ...] <i>file</i> ... int rand() int srand(<i>seed</i>); unsigned <i>seed</i> ;</pre> |
| DESCRIPTION | <p>rand() uses a multiplicative congruential random number generator with period 2^{32} to return successive pseudo-random numbers in the range from 0 to "$2^{31} - 1$."</p> <p>srand() can be called at any time to reset the random-number generator to a random starting point. The generator is initially seeded with a value of 1.</p> |
| SEE ALSO | drand48(3C) , rand(3C) , random(3C) |
| NOTES | <p>Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.</p> <p>The spectral properties of rand() leave a great deal to be desired. drand48(3C) and random(3C) provide much better, though more elaborate, random-number generators.</p> <p>The low bits of the numbers generated are not very random; use the middle bits. In particular the lowest bit alternates between 0 and 1.</p> |

| NAME | rand, srand, rand_r – simple random-number generator | | | | |
|--------------------|--|----------------|-----------------|----------|------------------|
| SYNOPSIS | <pre>#include <stdlib.h> int rand(void); void srand(unsigned int seed); int rand_r(unsigned int * seed);</pre> | | | | |
| DESCRIPTION | <p>The rand() function uses a multiplicative congruential random-number generator with period 2^{32} that returns successive pseudo-random numbers in the range of 0 to <code>RAND_MAX</code> (defined in <code><stdlib.h></code>).</p> <p>The srand() function uses the argument <i>seed</i> as a seed for a new sequence of pseudo-random numbers to be returned by subsequent calls to rand(). If srand() is then called with the same <i>seed</i> value, the sequence of pseudo-random numbers will be repeated. If rand() is called before any calls to srand() have been made, the same sequence will be generated as when srand() is first called with a <i>seed</i> value of 1.</p> <p>The rand_r() function has the same functionality as rand() except that a pointer to a seed <i>seed</i> must be supplied by the caller. The seed to be supplied is not the same seed as in srand().</p> | | | | |
| USAGE | The spectral properties of rand() are limited. The drand48(3C) function provides a better, more elaborate random-number generator. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>See NOTES below.</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | See NOTES below. |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | See NOTES below. | | | | |
| SEE ALSO | drand48(3C) , attributes(5) | | | | |
| NOTES | <p>The rand_r() function is as proposed in the POSIX.4a Draft #6 document, and is subject to change to be compliant with the standard when it is accepted.</p> <p>The rand() is unsafe in multithreaded applications. The rand_r() function is MT-Safe, and should be used instead. The srand() function is unsafe in multithreaded applications.</p> <p>When compiling multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the compile line. This flag should only be used in multithreaded applications.</p> | | | | |

| | |
|--------------------|--|
| NAME | random, srandom, initState, setState – pseudorandom number functions |
| SYNOPSIS | <pre>#include <stdlib.h> long random(void); void srandom(unsigned int seed); char * initState(unsigned int seed, char *state, size_t size); char * setState(const char * state);</pre> |
| DESCRIPTION | <p>The random() function uses a nonlinear additive feedback random-number generator employing a default state array size of 31 long integers to return successive pseudo-random numbers in the range from 0 to $2^{31} - 1$. The period of this random-number generator is approximately $16 \times (2^{31} - 1)$. The size of the state array determines the period of the random-number generator. Increasing the state array size increases the period.</p> <p>The srandom() function initializes the current state array using the value of <i>seed</i>.</p> <p>The random() and srandom() functions have (almost) the same calling sequence and initialization properties as rand() and srand() (see rand(3C)). The difference is that rand(3C) produces a much less random sequence—in fact, the low dozen bits generated by rand go through a cyclic pattern. All the bits generated by random() are usable. For example,</p> <pre>random() & 01</pre> <p>will produce a random binary value.</p> <p>Unlike srand(), srandom() does not return the old seed because the amount of state information used is much more than a single word. Two other routines are provided to deal with restarting/changing random number generators. With 256 bytes of state information, the period of the random-number generator is greater than 2^{69}.</p> <p>Like rand(3C), random() produces by default a sequence of numbers that can be duplicated by calling srandom() with 1 as the seed.</p> <p>The initState() and setState() functions handle restarting and changing random-number generators. The initState() function allows a state array, pointed to by the <i>state</i> argument, to be initialized for future use. The <i>size</i> argument, which specifies the size in bytes of the state array, is used by initState() to decide what type of random-number generator to use; the larger the state array, the more random the numbers. Values for the amount of state information are 8, 32, 64, 128, and 256 bytes. Other values greater than 8 bytes</p> |

are rounded down to the nearest one of these values. For values smaller than 8, **random()** uses a simple linear congruential random number generator. The *seed* argument specifies a starting point for the random-number sequence and provides for restarting at the same point. The **initstate()** function returns a pointer to the previous state information array.

If **initstate()** has not been called, then **random()** behaves as though **initstate()** had been called with *seed* = 1 and *size* = 128.

If **initstate()** is called with *size* < 8, then **random()** uses a simple linear congruential random number generator.

Once a state has been initialized, **setstate()** allows switching between state arrays. The array defined by the *state* argument is used for further random-number generation until **initstate()** is called or **setstate()** is called again. The **setstate()** function returns a pointer to the previous state array.

RETURN VALUES

The **random()** function returns the generated pseudo-random number.

The **srandom()** function returns no value.

Upon successful completion, **initstate()** and **setstate()** return a pointer to the previous state array. Otherwise, a null pointer is returned.

ERRORS

No errors are defined.

USAGE

After initialization, a state array can be restarted at a different point in one of two ways:

- The **initstate()** function can be used, with the desired seed, state array, and size of the array.
- The **setstate()** function, with the desired state, can be used, followed by **srandom()** with the desired seed. The advantage of using both of these functions is that the size of the state array does not have to be saved once it is initialized.

EXAMPLES

EXAMPLE 1 Example to initialize an array and pass it in to **initstate**.

```
/* Initialize an array and pass it in to initstate. */
static long state1[32] = {
    3,
    0x9a319039, 0x32d9c024, 0x9b663182, 0x5da1f342,
    0x7449e56b, 0xbeb1dbb0, 0xab5c5918, 0x946554fd,
    0x8c2e680f, 0xeb3d799f, 0xb11ee0b7, 0x2d436b86,
    0xda672e2a, 0x1588ca88, 0xe369735d, 0x904f35f7,
    0xd7158fd6, 0x6fa6f051, 0x616e6b96, 0xac94efdc,
    0xde3b81e0, 0xdf0a6fb5, 0xf103bc02, 0x48f340fb,
    0x36413f93, 0xc622c298, 0xf5a42ab8, 0x8a88d77b,
    0xf5ad9d0e, 0x8999220b, 0x27fb47b9
};
```

```

main() {
    unsigned seed;
    int n;
    seed = 1;
    n = 128;
    initstate(seed, state1, n);
    setstate(state1);
    printf("%d,random( );
}

```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|------------------|
| MT-Level | See NOTES below. |

SEE ALSO

drand48(3C), **rand(3C)**, **attributes(5)**

NOTES

The **random()** and **srandom()** functions are unsafe in multithreaded applications.

Use of these functions in multithreaded applications is unsupported.

The **random()** and **srandom()** functions operate at about two-thirds the speed of **rand(3C)**.

| | |
|--------------------|--|
| NAME | rcmd, rresvport, ruserok – routines for returning a stream to a remote command |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lsocket -lnsl [<i>library</i> ...]</pre> <p>int rcmd(char ** <i>ahost</i>, unsigned short <i>inport</i>, const char * <i>luser</i>, const char * <i>ruser</i>, const char * <i>cmd</i>, int * <i>fd2p</i>);</p> <p>int rresvport(int * <i>port</i>);</p> <p>int ruserok(const char * <i>rhost</i>, int <i>suser</i>, const char * <i>ruser</i>, const char * <i>luser</i>);</p> |
| DESCRIPTION | <p>rcmd() is a routine used by the super-user to execute a command on a remote machine using an authentication scheme based on reserved port numbers. rresvport() is a routine which returns a descriptor to a socket with an address in the privileged port space. ruserok() is a routine used by servers to authenticate clients requesting service with rcmd. All three functions are present in the same file and are used by the in.rshd(1M) server (among others).</p> <p>rcmd() looks up the host <i>*ahost</i> using gethostbyname(3N), returning -1 if the host does not exist. Otherwise <i>*ahost</i> is set to the standard name of the host and a connection is established to a server residing at the well-known Internet port <i>inport</i>.</p> <p>If the connection succeeds, a socket in the Internet domain of type <code>SOCK_STREAM</code> is returned to the caller, and given to the remote command as its standard input (file descriptor 0) and standard output (file descriptor 1). If <i>fd2p</i> is non-zero, then an auxiliary channel to a control process will be set up, and a descriptor for it will be placed in <i>*fd2p</i>. The control process will return diagnostic output from the command (file descriptor 2) on this channel, and will also accept bytes on this channel as signal numbers, to be forwarded to the process group of the command. If <i>fd2p</i> is 0, then the standard error (file descriptor 2) of the remote command will be made the same as its standard output and no provision is made for sending arbitrary signals to the remote</p> |

process, although you may be able to get its attention by using out-of-band data.

The protocol is described in detail in `in.rshd(1M)`.

The `rresvport()` routine is used to obtain a socket bound to a privileged port number. This socket is suitable for use by `rcmd()` and several other routines. Privileged Internet ports are those in the range 1 to 1023. Only the super-user is allowed to bind a socket to a privileged port number. The application must pass in `port`, which must be in the range 512 to 1023. The system first tries to bind to that port number. If it fails, it then tries to bind to port numbers less than `port` until either it succeeds or port number 512 is reached.

`ruserok()` takes a remote host's name, as returned by a `gethostbyaddr()` (see `gethostbyname(3N)`) routine, two user names and a flag indicating whether the local user's name is that of the super-user. It then checks the files `/etc/hosts.equiv` and possibly `.rhosts` in the local user's home directory to see if the request for service is allowed. 0 is returned if the machine name is listed in the `/etc/hosts.equiv` file, or the host and remote user name are found in the `.rhosts` file; otherwise `ruserok()` returns -1. If the super-user flag is 1, the checking of the `/etc/hosts.equiv` file is bypassed.

RETURN VALUES

`rcmd()` returns a valid socket descriptor on success. It returns -1 on error and prints a diagnostic message on the standard error.

`rresvport()` returns a valid, bound socket descriptor on success. It returns -1 on error with the global value `errno` set according to the reason for failure.

FILES

| | |
|-------------------------------|--------------------------------|
| <code>/etc/hosts.equiv</code> | system trusted hosts and users |
| <code>~/.rhosts</code> | user's trusted hosts and users |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`rlogin(1)`, `rsh(1)`, `in.rexecd(1M)`, `in.rshd(1M)`, `intro(2)`, `gethostbyname(3N)`, `rexec(3N)`, `attributes(5)`

NOTES

The error code EAGAIN is overloaded to mean "All network ports in use."

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

| | |
|----------------------|--|
| NAME | readdir – read a directory entry |
| SYNOPSIS | <pre> /usr/ucb/cc [flag ...] file ... #include <sys/types.h> #include <sys/dir.h> struct direct *readdir(<i>dirp</i>); DIR *dirp; </pre> |
| DESCRIPTION | <p>The readdir() function returns a pointer to a structure representing the directory entry at the current position in the directory stream to which <i>dirp</i> refers, and positions the directory stream at the next entry, except on read-only file systems. It returns a <code>NULL</code> pointer upon reaching the end of the directory stream, or upon detecting an invalid location in the directory. The readdir() function shall not return directory entries containing empty names. It is unspecified whether entries are returned for dot (<code>.</code>) or dot-dot (<code>..</code>). The pointer returned by readdir() points to data that may be overwritten by another call to readdir() on the same directory stream. This data shall not be overwritten by another call to readdir() on a different directory stream. The readdir() function may buffer several directory entries per actual read operation. The readdir() function marks for update the <i>st_atime</i> field of the directory each time the directory is actually read.</p> |
| RETURN VALUES | The readdir() function returns <code>NULL</code> on failure and sets <code>errno</code> to indicate the error. |
| ERRORS | <p>The readdir() function will fail if one or more of the following are true:</p> <p>EAGAIN Mandatory file/record locking was set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set, and there was a blocking record lock.</p> <p>EAGAIN Total amount of system memory available when reading using raw I/O is temporarily insufficient.</p> <p>EAGAIN No data is waiting to be read on a file associated with a tty device and <code>O_NONBLOCK</code> was set.</p> <p>EAGAIN No message is waiting to be read on a stream and <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set.</p> <p>EBADF The file descriptor determined by the <code>DIR</code> stream is no longer valid. This results if the <code>DIR</code> stream has been closed.</p> <p>EBADMSG Message waiting to be read on a stream is not a data message.</p> <p>EDEADLK The read() was going to go to sleep and cause a deadlock to occur.</p> |

| | |
|-------------------|---|
| EFAULT | <i>buf</i> points to an illegal address. |
| EINTR | A signal was caught during the read() or readv() function. |
| EINVAL | Attempted to read from a stream linked to a multiplexor. |
| EIO | A physical I/O error has occurred, or the process is in a background process group and is attempting to read from its controlling terminal, and either the process is ignoring or blocking the SIGTTIN signal or the process group of the process is orphaned. |
| ENOENT | The current file pointer for the directory is not located at a valid entry. |
| ENOLCK | The system record lock table was full, so the read() or readv() could not go to sleep until the blocking record lock was removed. |
| ENOLINK | <i>fildev</i> is on a remote machine and the link to that machine is no longer active. |
| ENXIO | The device associated with <i>fildev</i> is a block special or character special file and the value of the file pointer is out of range. |
| E_OVERFLOW | The value of the <code>direct</code> structure member <code>d_ino</code> cannot be represented in an <code>ino_t</code> . |

USAGE The **readdir()** function has a transitional interface for 64-bit file offsets. See **1f64(5)**.

SEE ALSO **getdents(2)**, **readdir(3C)**, **scandir(3B)**, **1f64(5)**

NOTES Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

| | |
|--------------------|---|
| NAME | readdir, readdir_r – read directory |
| SYNOPSIS | <pre>#include <sys/types.h> #include <dirent.h> struct dirent * readdir(DIR * <i>dirp</i>); struct dirent * readdir_r(DIR * <i>dirp</i>, struct dirent * <i>entry</i>);</pre> |
| POSIX | <pre>cc [<i>flag</i> ...] <i>file</i> ... -D_POSIX_PTHREAD_SEMANTICS [<i>library</i> ...] int readdir_r(DIR * <i>dirp</i>, struct dirent * <i>entry</i>, struct dirent ** <i>result</i>);</pre> |
| DESCRIPTION | <p>The type <code>DIR</code>, which is defined in the header <code><dirent.h></code>, represents a <i>directory stream</i>, which is an ordered sequence of all the directory entries in a particular directory. Directory entries represent files; files may be removed from a directory or added to a directory asynchronously to the operation of <code>readdir()</code> and <code>readdir_r()</code>.</p> <p>readdir()</p> <p>The <code>readdir()</code> function returns a pointer to a structure representing the directory entry at the current position in the directory stream specified by the argument <code>dirp</code>, and positions the directory stream at the next entry. It returns a null pointer upon reaching the end of the directory stream. The structure <code>dirent</code> defined by the <code><dirent.h></code> header describes a directory entry.</p> <p>If entries for dot or dot-dot exist, one entry will be returned for dot and one entry will be returned for dot-dot; otherwise they will not be returned.</p> <p>The pointer returned by <code>readdir()</code> points to data which may be overwritten by another call to <code>readdir()</code> on the same directory stream. This data is not overwritten by another call to <code>readdir()</code> on a different directory stream.</p> <p>If a file is removed from or added to the directory after the most recent call to <code>opendir(3C)</code> or <code>rewinddir(3C)</code>, whether a subsequent call to <code>readdir()</code> returns an entry for that file is unspecified.</p> <p>The <code>readdir()</code> function may buffer several directory entries per actual read operation; <code>readdir()</code> marks for update the <code>st_atime</code> field of the directory each time the directory is actually read.</p> |

After a call to `fork(2)`, either the parent or child (but not both) may continue processing the directory stream using `readdir()`, `rewinddir()` or `seekdir(3C)`. If both the parent and child processes use these functions, the result is undefined.

If the entry names a symbolic link, the value of the `d_ino` member is unspecified.

readdir_r()

The `readdir_r()` function is equivalent to `readdir()` except that a buffer *result* must be supplied by the caller to store the result. The size should be `sizeof(struct dirent) + {NAME_MAX}` (that is, `pathconf(_PC_NAME_MAX) + 1`). `_PC_NAME_MAX` is defined in `<unistd.h>`.

The POSIX version (see `standards(5)`) of the `readdir_r()` function initializes the structure referenced by *entry* and stores a pointer to this structure in *result*.

RETURN VALUES

Upon successful completion, `readdir()` and `readdir_r()` return a pointer to an object of type `struct dirent`. When an error is encountered, a null pointer is returned and `errno` is set to indicate the error. When the end of the directory is encountered, a null pointer is returned and `errno` is not changed. The POSIX `readdir_r()` returns 0 if successful or an error number to indicate failure.

ERRORS

The `readdir()` function will fail if:

E_OVERFLOW One of the values in the structure to be returned cannot be represented correctly.

The `readdir()` and `readdir_r()` functions will fail if:

EBADF The file descriptor determined by the `DIR` stream is no longer valid. This results if the `DIR` stream has been closed.

ENOENT The current file pointer for the directory is not located at a valid entry.

The `readdir()` and `readdir_r()` functions may fail if:

EBADF The *dirp* argument does not refer to an open directory stream.

ENOENT The current position of the directory stream is invalid.

USAGE

The `readdir()` function should be used in conjunction with `opendir()`, `closedir()`, and `rewinddir()` to examine the contents of the directory. As `readdir()` returns a null pointer both at the end of the directory and on error, an application wishing to check for error situations should set `errno` to 0, then call `readdir()`, then check `errno` and if it is non-zero, assume an error has occurred.

The **readdir()** and **readdir_r()** functions have transitional interfaces for 64-bit file offsets. See **1f64(5)**.

EXAMPLES

EXAMPLE 1 Example of sample code that will search the current directory for the entry *name*.

The following sample code will search the current directory for the entry *name*:

```
dirp = opendir(".");
while ((dp = readdir(dirp)) != NULL)
    if (strcmp(dp->d_name, name) == 0) {
        closedir(dirp);
        return FOUND;
    }
closedir(dirp);
return NOT_FOUND;
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | See NOTES below. |

SEE ALSO

fork(2), **lstat(2)**, **symlink(2)**, **Intro(3)**, **closedir(3C)**, **opendir(3C)**, **rewinddir(3C)**, **seekdir(3C)**, **attributes(5)**, **1f64(5)**, **standards(5)**

NOTES

When compiling multithreaded programs, see **Intro(3)**, *Notes On Multithreaded Applications*.

The **readdir()** function is unsafe in multithreaded applications. The **readdir_r()** function is safe, and should be used instead.

Solaris 2.4 and earlier releases provided a **readdir_r()** interface as specified in POSIX.1c Draft 6. The final POSIX.1c standard changed the interface as described above. Support for the Draft 6 interface is provided for compatibility only and may not be supported in future releases. New applications and libraries should use the POSIX standard interface.

For POSIX.1c-compliant applications, the **_POSIX_PTHREAD_SEMANTICS** and **_REENTRANT** flags are automatically turned on by defining the **_POSIX_C_SOURCE** flag with a value $\geq 199506L$.

| | |
|----------------------|---|
| NAME | read_vtoc, write_vtoc – read and write a disk's VTOC |
| SYNOPSIS | <pre>#include <sys/vtoc.h> cc [<i>flag</i> ...] <i>file</i> ... -ladm [<i>library</i> ...] int read_vtoc(int <i>fd</i>, struct vtoc * <i>vtoc</i>); int write_vtoc(int <i>fd</i>, struct vtoc * <i>vtoc</i>);</pre> |
| DESCRIPTION | <p>read_vtoc() returns the VTOC (volume table of contents) structure that is stored on the disk associated with the open file descriptor <i>fd</i> .</p> <p>write_vtoc() stores the VTOC structure on the disk associated with the open file descriptor <i>fd</i> .</p> <p><i>fd</i> refers to any slice on a raw disk.</p> |
| RETURN VALUES | <pre>read_vtoc returns: positive number Success. The positive number is the slice index associated with the open file descriptor. negative number There are two possible error returns. VT_EIO indicates an I/O error occurred and VT_ERROR indicates an unknown error. write_vtoc returns: 0 Success negative number There are three possible error returns. VT_EIO indicates an I/O error occurred, VT_ERROR indicates an unknown error, and VT_EINVAL indicates an incorrect field within the VTOC.</pre> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO `fmthard(1M)`, `format(1M)`, `prtvtoc(1M)`, `ioctl(2)`, `attributes(5)`, `dkio(7I)`

BUGS `write_vtoc` cannot write a VTOC on an unlabeled disk. Use `format(1M)` for this purpose.

| | | | | | | | | | | | | | | | | | |
|----------------------|---|---------------|--|---------------|---|------------|---|--------------|---|---------------------|---|---------------|---|----------------|--|---------------------|---|
| NAME | realpath – resolve pathname | | | | | | | | | | | | | | | | |
| SYNOPSIS | <pre>#include <stdlib.h> char *realpath(const char *file_name, char *resolved_name);</pre> | | | | | | | | | | | | | | | | |
| DESCRIPTION | <p>The realpath() function derives, from the pathname pointed to by <i>file_name</i>, an absolute pathname that names the same file, whose resolution does not involve ".", ". .", or symbolic links. The generated pathname is stored, up to a maximum of <code>PATH_MAX</code> bytes, in the buffer pointed to by <i>resolved_name</i>.</p> <p>The realpath() function can handle both relative and absolute path names. For absolute path names and the relative names whose resolved name cannot be expressed relatively (for example, <code>./../reldir</code>), it returns the <i>resolved absolute</i> name. For the other relative path names, it returns the <i>resolved relative</i> name.</p> | | | | | | | | | | | | | | | | |
| RETURN VALUES | On successful completion, realpath() returns a pointer to the resolved name. Otherwise, realpath() returns a null pointer and sets <code>errno</code> to indicate the error, and the contents of the buffer pointed to by <i>resolved_name</i> are undefined. | | | | | | | | | | | | | | | | |
| ERRORS | <p>The realpath() function will fail if:</p> <table border="0"> <tr> <td style="vertical-align: top;">EACCES</td> <td>Read or search permission was denied for a component of <i>file_name</i>.</td> </tr> <tr> <td style="vertical-align: top;">EINVAL</td> <td>Either the <i>file_name</i> or <i>resolved_name</i> argument is a null pointer.</td> </tr> <tr> <td style="vertical-align: top;">EIO</td> <td>An error occurred while reading from the file system.</td> </tr> <tr> <td style="vertical-align: top;">ELOOP</td> <td>Too many symbolic links were encountered in resolving <i>path</i>.</td> </tr> <tr> <td style="vertical-align: top;">ENAMETOOLONG</td> <td>The <i>file_name</i> argument is longer than <code>PATH_MAX</code> or a pathname component is longer than <code>NAME_MAX</code>.</td> </tr> <tr> <td style="vertical-align: top;">ENOENT</td> <td>A component of <i>file_name</i> does not name an existing file or <i>file_name</i> points to an empty string.</td> </tr> <tr> <td style="vertical-align: top;">ENOTDIR</td> <td>A component of the path prefix is not a directory.</td> </tr> </table> <p>The realpath() function may fail if:</p> <table border="0"> <tr> <td style="vertical-align: top;">ENAMETOOLONG</td> <td>Pathname resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code>.</td> </tr> </table> | EACCES | Read or search permission was denied for a component of <i>file_name</i> . | EINVAL | Either the <i>file_name</i> or <i>resolved_name</i> argument is a null pointer. | EIO | An error occurred while reading from the file system. | ELOOP | Too many symbolic links were encountered in resolving <i>path</i> . | ENAMETOOLONG | The <i>file_name</i> argument is longer than <code>PATH_MAX</code> or a pathname component is longer than <code>NAME_MAX</code> . | ENOENT | A component of <i>file_name</i> does not name an existing file or <i>file_name</i> points to an empty string. | ENOTDIR | A component of the path prefix is not a directory. | ENAMETOOLONG | Pathname resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code> . |
| EACCES | Read or search permission was denied for a component of <i>file_name</i> . | | | | | | | | | | | | | | | | |
| EINVAL | Either the <i>file_name</i> or <i>resolved_name</i> argument is a null pointer. | | | | | | | | | | | | | | | | |
| EIO | An error occurred while reading from the file system. | | | | | | | | | | | | | | | | |
| ELOOP | Too many symbolic links were encountered in resolving <i>path</i> . | | | | | | | | | | | | | | | | |
| ENAMETOOLONG | The <i>file_name</i> argument is longer than <code>PATH_MAX</code> or a pathname component is longer than <code>NAME_MAX</code> . | | | | | | | | | | | | | | | | |
| ENOENT | A component of <i>file_name</i> does not name an existing file or <i>file_name</i> points to an empty string. | | | | | | | | | | | | | | | | |
| ENOTDIR | A component of the path prefix is not a directory. | | | | | | | | | | | | | | | | |
| ENAMETOOLONG | Pathname resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code> . | | | | | | | | | | | | | | | | |

ENOMEM Insufficient storage space is available.

USAGE The **realpath()** function operates on null-terminated strings.

One should have execute permission on all the directories in the given and the resolved path.

The **realpath()** function may fail to return to the current directory if an error occurs.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **getcwd(3C)**, **sysconf(3C)**, **attributes(5)**

| | |
|----------------------|--|
| NAME | reboot – reboot system or halt processor |
| SYNOPSIS | <pre>#include <sys/reboot.h> int reboot(int howto, char *bootargs);</pre> |
| DESCRIPTION | <p>The reboot() function reboots the system. The <i>howto</i> argument specifies the behavior of the system while rebooting and can assume one of the following values:</p> <p>RE_AUTOBOOT The machine is rebooted from the root filesystem on the default boot device. This is the default behavior. See boot(1M) and kernel(1M).</p> <p>RB_HALT the processor is simply halted; no reboot takes place. This option should be used with caution.</p> <p>RB_ASKNAME Interpreted by the bootstrap program and kernel, causing the user to be asked for pathnames during the bootstrap.</p> <p>Any other <i>howto</i> argument causes the kernel file to boot.</p> <p>The interpretation of the <i>bootargs</i> argument is platform-dependent.</p> |
| RETURN VALUES | Upon successful completion, reboot() never returns. Otherwise, <code>-1</code> is returned and <code>errno</code> is set to indicate the error. |
| ERRORS | The reboot() function will fail if: EPERM The caller is not the super-user. |
| USAGE | Only the super-user may reboot() a machine. |
| SEE ALSO | intro(1M) , boot(1M) , halt(1M) , init(1M) , kernel(1M) , reboot(1M) , uadmin(2) |

| | |
|----------------------|--|
| NAME | re_comp, re_exec – compile and execute regular expressions |
| SYNOPSIS | <pre>#include <re_comp.h> char * re_comp(const char * string); int re_exec(const char * string);</pre> |
| DESCRIPTION | <p>The re_comp() function converts a regular expression string (RE) into an internal form suitable for pattern matching. The re_exec() function compares the string pointed to by the <i>string</i> argument with the last regular expression passed to re_comp() .</p> <p>If re_comp() is called with a null pointer argument, the current regular expression remains unchanged.</p> <p>Strings passed to both re_comp() and re_exec() must be terminated by a null byte, and may include NEWLINE characters.</p> <p>The re_comp() and re_exec() functions support <i>simple regular expressions</i> , which are defined on the regexp(5) manual page. The regular expressions of the form <code>\\{ m \\}</code> , <code>\\{ m, \\}</code> , or <code>\\{ m,n \\}</code> are not supported.</p> |
| RETURN VALUES | <p>The re_comp() function returns a null pointer when the string pointed to by the <i>string</i> argument is successfully converted. Otherwise, a pointer to one of the following error message strings is returned:</p> <pre>No previous regular expression Regular expression too long unmatched \\(missing too many \\(\\) pairs unmatched \\)</pre> <p>Upon successful completion, re_exec() returns 1 if <i>string</i> matches the last compiled regular expression. Otherwise, re_exec() returns 0 if <i>string</i> fails to match the last compiled regular expression, and -1 if the compiled regular expression is invalid (indicating an internal error).</p> |
| ERRORS | No errors are defined. |
| USAGE | For portability to implementations conforming to X/Open standards prior to SUS, regcomp(3C) and regex(3C) are preferred to these functions. See standards(5) . |
| SEE ALSO | grep(1) , regcmp(1) , regcmp(3C) , regcomp(3C) , regex(3C) , regexpr(3G) , regexp(5) , standards(5) |

| | | | |
|--------------------|---|---------|---|
| NAME | recv, recvfrom, recvmsg – receive a message from a socket | | |
| SYNOPSIS | <pre> cc [flag ...] file ... -lsocket -lnsl [library ...] #include <sys/types.h> #include <sys/socket.h> #include <sys/uio.h> ssize_t recv(int s, void * buf, size_t len, int flags); ssize_t recvfrom(int s, void * buf, size_t len, int flags, struct sockaddr * from, socklen_t * fromlen); ssize_t recvmsg(int s, struct msghdr * msg, int flags); </pre> | | |
| DESCRIPTION | <p>recv() , recvfrom() , and recvmsg() are used to receive messages from another socket. recv() may be used only on a <i>connected</i> socket (see connect(3N)), while recvfrom() and recvmsg() may be used to receive data on a socket whether it is in a connected state or not. <i>s</i> is a socket created with socket(3N) .</p> <p>If <i>from</i> is not a NULL pointer, the source address of the message is filled in. <i>fromlen</i> is a value-result parameter, initialized to the size of the buffer associated with <i>from</i> , and modified on return to indicate the actual size of the address stored there. The length of the message is returned. If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket the message is received from (see socket(3N)).</p> <p>If no messages are available at the socket, the receive call waits for a message to arrive, unless the socket is nonblocking (see fcntl(2)) in which case -1 is returned with the external variable errno set to EWOULDBLOCK .</p> <p>The select() call may be used to determine when more data arrives.</p> <p>The <i>flags</i> parameter is formed by ORing one or more of the following:</p> <table border="0" style="margin-left: 20px;"> <tr> <td style="padding-right: 20px;">MSG_OOB</td> <td>Read any “out-of-band” data present on the socket rather than the regular “in-band” data.</td> </tr> </table> | MSG_OOB | Read any “out-of-band” data present on the socket rather than the regular “in-band” data. |
| MSG_OOB | Read any “out-of-band” data present on the socket rather than the regular “in-band” data. | | |

MSG_PEEK “Peek” at the data present on the socket; the data is returned, but not consumed, so that a subsequent receive operation will see the same data.

The **recvmsg()** call uses a `msg_hdr` structure to minimize the number of directly supplied parameters. This structure is defined in `<sys/socket.h>` and includes the following members:

```

caddr_t      msg_name;          /* optional address */
int          msg_namelen;      /* size of address */
struct iovec *msg_iov;        /* scatter/gather array */
int          msg_iovlen;      /* # elements in msg_iov */
caddr_t      msg_accrights;    /* access rights sent/received */
int          msg_accrightslen;

```

Here `msg_name` and `msg_namelen` specify the destination address if the socket is unconnected; `msg_name` may be given as a NULL pointer if no names are desired or required. The `msg_iov` and `msg_iovlen` describe the scatter-gather locations, as described in `read(2)`. A buffer to receive any access rights sent along with the message is specified in `msg_accrights`, which has length `msg_accrightslen`.

RETURN VALUES

These calls return the number of bytes received, or `-1` if an error occurred.

ERRORS

The calls fail if:

| | |
|-----------------|---|
| EBADF | <code>s</code> is an invalid file descriptor. |
| EINTR | The operation was interrupted by delivery of a signal before any data was available to be received. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| ENOMEM | There was insufficient user memory available for the operation to complete. |
| ENOSR | There were insufficient STREAMS resources available for the operation to complete. |
| ENOTSOCK | <code>s</code> is not a socket. |
| ESTALE | A stale NFS file handle exists. |

EWouldBlock The socket is marked non-blocking and the requested operation would block.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

`fcntl(2)`, `ioctl(2)`, `read(2)`, `connect(3N)`, `getsockopt(3N)`, `send(3N)`, `socket(3N)`, `attributes(5)`, `socket(5)`

| | |
|--------------------|--|
| NAME | recv – receive a message from a connected socket |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lxnet [<i>library</i> ...] #include <sys/socket.h></pre> <p>ssize_t recv(int <i>socket</i>, void *<i>buffer</i>, size_t <i>length</i>, int <i>flags</i>);</p> |
| DESCRIPTION | <p>The recv() function receives a message from a connection-mode or connectionless-mode socket. It is normally used with connected sockets because it does not permit the application to retrieve the source address of received data. The function takes the following arguments:</p> <p>socket Specifies the socket file descriptor.</p> <p>buffer Points to a buffer where the message should be stored.</p> <p>length Specifies the length in bytes of the buffer pointed to by the <i>buffer</i> argument.</p> <p>flags Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:</p> <p>MSG_PEEK Peeks at an incoming message. The data is treated as unread and the next recv() or similar function will still return this data.</p> <p>MSG_OOB Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.</p> <p>MSG_WAITALL Requests that the function block until the full amount of data requested can be returned. The function may return a smaller amount of data if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket.</p> <p>The recv() function returns the length of the message written to the buffer pointed to by the <i>buffer</i> argument. For message-based sockets such as SOCK_DGRAM and SOCK_SEQPACKET, the entire message must be read in a</p> |

single operation. If a message is too long to fit in the supplied buffer, and MSG_PEEK is not set in the *flags* argument, the excess bytes are discarded. For stream-based sockets such as SOCK_STREAM, message boundaries are ignored. In this case, data is returned to the user as soon as it becomes available, and no data is discarded.

If the MSG_WAITALL flag is not set, data will be returned only up to the end of the first message.

If no messages are available at the socket and O_NONBLOCK is not set on the socket's file descriptor, `recv()` blocks until a message arrives. If no messages are available at the socket and O_NONBLOCK is set on the socket's file descriptor, `recv()` fails and sets `errno` to EAGAIN or EWOULDBLOCK.

USAGE

The `recv()` function is identical to `recvfrom(3XN)` with a zero *address_len* argument, and to `read()` if no flags are used.

The `select(3C)` and `poll(2)` functions can be used to determine when data is available to be received.

RETURN VALUES

Upon successful completion, `recv()` returns the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, `recv()` returns 0. Otherwise, -1 is returned and `errno` is set to indicate the error.

ERRORS

The `recv()` function will fail if:

EAGAIN**EWOULDBLOCK**

The socket's file descriptor is marked O_NONBLOCK and no data is waiting to be received; or MSG_OOB is set and no out-of-band data is available and either the socket's file descriptor is marked O_NONBLOCK or the socket does not support blocking to await out-of-band data.

EBADF

The *socket* argument is not a valid file descriptor.

ECONNRESET

A connection was forcibly closed by a peer.

EFAULT

The *buffer* parameter can not be accessed or written.

EINTR

The `recv()` function was interrupted by a signal that was caught, before any data was available.

EINVAL

The MSG_OOB flag is set and no out-of-band data is available.

| | |
|---|--|
| ENOTCONN | A receive is attempted on a connection-mode socket that is not connected. |
| ENOTSOCK | The <i>socket</i> argument does not refer to a socket. |
| EOPNOTSUPP | The specified flags are not supported for this socket type or protocol. |
| ETIMEDOUT | The connection timed out during connection establishment, or due to a transmission timeout on active connection. |
| The recv() function may fail if: | |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| ENOBUFS | Insufficient resources were available in the system to perform the operation. |
| ENOMEM | Insufficient memory was available to fulfill the request. |
| ENOSR | There were insufficient STREAMS resources available for the operation to complete. |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

poll(2), **recvmsg(3XN)**, **recvfrom(3XN)**, **select(3C)**, **send(3XN)**, **sendmsg(3XN)**, **sendto(3XN)**, **shutdown(3XN)**, **socket(3XN)**, **attributes(5)**

| | |
|--------------------|--|
| NAME | recvfrom – receive a message from a socket |
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lxnet [<i>library ...</i>] #include <sys/socket.h></pre> <p>ssize_t recvfrom(int <i>socket</i>, void *<i>buffer</i>, size_t <i>length</i>, int <i>flags</i>, struct sockaddr *<i>address</i>, socklen_t *<i>address_len</i>);</p> |
| DESCRIPTION | <p>The recvfrom() function receives a message from a connection-mode or connectionless-mode socket. It is normally used with connectionless-mode sockets because it permits the application to retrieve the source address of received data.</p> <p>The function takes the following arguments:</p> <p>socket Specifies the socket file descriptor.</p> <p>buffer Points to the buffer where the message should be stored.</p> <p>length Specifies the length in bytes of the buffer pointed to by the <i>buffer</i> argument.</p> <p>flags Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:</p> <p style="margin-left: 40px;">MSG_PEEK Peeks at an incoming message. The data is treated as unread and the next recvfrom() or similar function will still return this data.</p> <p style="margin-left: 40px;">MSG_OOB Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.</p> <p style="margin-left: 40px;">MSG_WAITALL Requests that the function block until the full amount of data requested can be returned. The function may return a smaller amount of data if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if</p> |

an error is pending for the socket.

address A null pointer, or points to a `sockaddr` structure in which the sending address is to be stored. The length and format of the address depend on the address family of the socket.

address_len Specifies the length of the `sockaddr` structure pointed to by the *address* argument.

The `recvfrom()` function returns the length of the message written to the buffer pointed to by the *buffer* argument. For message-based sockets such as `SOCK_DGRAM` and `SOCK_SEQPACKET`, the entire message must be read in a single operation. If a message is too long to fit in the supplied buffer, and `MSG_PEEK` is not set in the *flags* argument, the excess bytes are discarded. For stream-based sockets such as `SOCK_STREAM`, message boundaries are ignored. In this case, data is returned to the user as soon as it becomes available, and no data is discarded.

If the `MSG_WAITALL` flag is not set, data will be returned only up to the end of the first message.

Not all protocols provide the source address for messages. If the *address* argument is not a null pointer and the protocol provides the source address of messages, the source address of the received message is stored in the `sockaddr` structure pointed to by the *address* argument, and the length of this address is stored in the object pointed to by the *address_len* argument.

If the actual length of the address is greater than the length of the supplied `sockaddr` structure, the stored address will be truncated.

If the *address* argument is not a null pointer and the protocol does not provide the source address of messages, the the value stored in the object pointed to by *address* is unspecified.

If no messages are available at the socket and `O_NONBLOCK` is not set on the socket's file descriptor, `recvfrom()` blocks until a message arrives. If no messages are available at the socket and `O_NONBLOCK` is set on the socket's file descriptor, `recvfrom()` fails and sets `errno` to `EAGAIN` or `EWOULDBLOCK`.

USAGE

The `select(3C)` and `poll(2)` functions can be used to determine when data is available to be received.

RETURN VALUES

Upon successful completion, `recvfrom()` returns the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, `recvfrom()` returns 0. Otherwise the function returns `-1` and sets `errno` to indicate the error.

ERRORS

The **recvfrom()** function will fail if:

EAGAIN**EWOULDBLOCK**

The socket's file descriptor is marked `O_NONBLOCK` and no data is waiting to be received; or `MSG_OOB` is set and no out-of-band data is available and either the socket's file descriptor is marked `O_NONBLOCK` or the socket does not support blocking to await out-of-band data.

EBADF

The *socket* argument is not a valid file descriptor.

ECONNRESET

A connection was forcibly closed by a peer.

EFAULT

The *buffer*, *address* or *address_len* parameter can not be accessed or written.

EINTR

A signal interrupted **recvfrom()** before any data was available.

EINVAL

The `MSG_OOB` flag is set and no out-of-band data is available.

ENOTCONN

A receive is attempted on a connection-mode socket that is not connected.

ENOTSOCK

The *socket* argument does not refer to a socket.

EOPNOTSUPP

The specified flags are not supported for this socket type.

ETIMEDOUT

The connection timed out during connection establishment, or due to a transmission timeout on active connection.

The **recvfrom()** function may fail if:

EIO

An I/O error occurred while reading from or writing to the file system.

ENOBUFS

Insufficient resources were available in the system to perform the operation.

ENOMEM

Insufficient memory was available to fulfill the request.

ENOSR

There were insufficient STREAMS resources available for the operation to complete.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

poll(2), **recv(3XN)**, **recvmsg(3XN)**, **select(3C)**, **send(3XN)**, **sendmsg(3XN)**, **sendto(3XN)**, **shutdown(3XN)**, **socket(3XN)**, **attributes(5)**

| | |
|--------------------|---|
| NAME | recvmsg – receive a message from a socket |
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lxnet [<i>library ...</i>] #include <sys/socket.h></pre> <p> ssize_t recvmsg(int <i>socket</i>, struct msghdr *<i>message</i>, int <i>flags</i>);</p> |
| DESCRIPTION | <p>The recvmsg() function receives a message from a connection-mode or connectionless-mode socket. It is normally used with connectionless-mode sockets because it permits the application to retrieve the source address of received data.</p> <p>The function takes the following arguments:</p> <p>socket Specifies the socket file descriptor.</p> <p>message Points to a <code>msghdr</code> structure, containing both the buffer to store the source address and the buffers for the incoming message. The length and format of the address depend on the address family of the socket. The <code>msg_flags</code> member is ignored on input, but may contain meaningful values on output.</p> <p>flags Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:</p> <p style="margin-left: 40px;">MSG_OOB Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.</p> <p style="margin-left: 40px;">MSG_PEEK Peeks at the incoming message.</p> <p style="margin-left: 40px;">MSG_WAITALL Requests that the function block until the full amount of data requested can be returned. The function may return a smaller amount of data if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if MSG_WAITALL was specified.</p> <p>The recvmsg() function receives messages from unconnected or connected sockets and returns the length of the message. socket.</p> |

The **recvmsg()** function returns the total length of the message. For message-based sockets such as SOCK_DGRAM and SOCK_SEQPACKET, the entire message must be read in a single operation. If a message is too long to fit in the supplied buffers, and MSG_PEEK is not set in the *flags* argument, the excess bytes are discarded, and MSG_TRUNC is set in the *msg_flags* member of the *msg_hdr* structure. For stream-based sockets such as SOCK_STREAM, message boundaries are ignored. In this case, data is returned to the user as soon as it becomes available, and no data is discarded.

If the MSG_WAITALL flag is not set, data will be returned only up to the end of the first message.

If no messages are available at the socket and O_NONBLOCK is not set on the socket's file descriptor, **recvfrom(3XN)** blocks until a message arrives. If no messages are available at the socket and O_NONBLOCK is set on the socket's file descriptor, **recvfrom(3XN)** function fails and sets *errno* to EAGAIN or EWOULDBLOCK.

In the *msg_hdr* structure, the *msg_name* and *msg_namelen* members specify the source address if the socket is unconnected. If the socket is connected, the *msg_name* and *msg_namelen* members are ignored. The *msg_name* member may be a null pointer if no names are desired or required. The *msg_iov* and *msg_iovlen* fields are used to specify where the received data will be stored. *msg_iov* points to an array of *iovec* structures; *msg_iovlen* must be set to the dimension of this array. In each *iovec* structure, the *iov_base* field specifies a storage area and the *iov_len* field gives its size in bytes. Each storage area indicated by *msg_iov* is filled with received data in turn until all of the received data is stored or all of the areas have been filled.

On successful completion, the *msg_flags* member of the message header is the bitwise-inclusive OR of all of the following flags that indicate conditions detected for the received message:

| | |
|-------------------|--|
| MSG_EOR | End of record was received (if supported by the protocol). |
| MSG_OOB | Out-of-band data was received. |
| MSG_TRUNC | Normal data was truncated. |
| MSG_CTRUNC | Control data was truncated. |

USAGE

The **select(3C)** and **poll(2)** functions can be used to determine when data is available to be received.

RETURN VALUES

Upon successful completion, **recvmsg()** returns the length of the message in bytes. If no messages are available to be received and the peer has performed

an orderly shutdown, **recvmsg()** returns 0. Otherwise, -1 is returned and **errno** is set to indicate the error.

ERRORS

The **recvmsg()** function will fail if:

EAGAIN**EWOULDBLOCK**

The socket's file descriptor is marked **O_NONBLOCK** and no data is waiting to be received; or **MSG_OOB** is set and no out-of-band data is available and either the socket's file descriptor is marked **O_NONBLOCK** or the socket does not support blocking to await out-of-band data.

EBADF

The *socket* argument is not a valid open file descriptor.

ECONNRESET

A connection was forcibly closed by a peer.

EFAULT

The *message* parameter, or storage pointed to by the *msg_name*, *msg_control* or *msg_iov* fields of the *message* parameter, or storage pointed to by the *iovec* structures pointed to by the *msg_iov* field can not be accessed or written.

EINTR

This function was interrupted by a signal before any data was available.

EINVAL

The sum of the *iov_len* values overflows an *ssize_t*. or the **MSG_OOB** flag is set and no out-of-band data is available.

EMSGSIZE

The *msg_iovlen* member of the *msg_hdr* structure pointed to by *message* is less than or equal to 0, or is greater than **IOV_MAX**.

ENOTCONN

A receive is attempted on a connection-mode socket that is not connected.

ENOTSOCK

The *socket* argument does not refer to a socket.

EOPNOTSUPP

The specified flags are not supported for this socket type.

ETIMEDOUT

The connection timed out during connection establishment, or due to a transmission timeout on active connection.

The **recvmsg()** function may fail if:

- EIO** An IO error occurred while reading from or writing to the file system.
- ENOBUFS** Insufficient resources were available in the system to perform the operation.
- ENOMEM** Insufficient memory was available to fulfill the request.
- ENOSR** There were insufficient STREAMS resources available for the operation to complete.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

poll(2), **recv(3XN)**, **recvfrom(3XN)**, **select(3C)**, **send(3XN)**, **sendmsg(3XN)**, **sendto(3XN)**, **shutdown(3XN)**, **socket(3XN)**, **attributes(5)**

| | |
|----------------------|---|
| NAME | redrawwin, wredrawln – redraw screen or portion of screen |
| SYNOPSIS | <pre>#include <curses.h> int redrawwin(WINDOW * win); int wredrawln(WINDOW * win, int beg_line, int num_lines);</pre> |
| PARAMETERS | <p><i>win</i> Is a pointer to the window in which to redraw.</p> <p><i>beg_line</i> Is the first line to redraw.</p> <p><i>num_lines</i> Is the number of lines to redraw.</p> |
| DESCRIPTION | <p>The redrawwin() and wredrawln() functions force portions of a window to be redrawn to the terminal when the next refresh operation is performed.</p> <p>The redrawwin() function forces the entire window <i>win</i> to be redrawn, while the wredrawln() function forces only <i>num_lines</i> lines starting with <i>beg_line</i> to be redrawn. Normally, refresh operations use optimization methods to reduce the actual amount of the screen to redraw based on the current screen contents. These functions tell the refresh operations not to attempt any optimization when redrawing the indicated areas.</p> <p>These functions are useful when the data that exists on the screen is believed to be corrupt and for applications such as screen editors that redraw portions of the screen.</p> |
| RETURN VALUES | On success, these functions return <code>OK</code> . Otherwise, they return <code>ERR</code> . |
| ERRORS | None. |
| SEE ALSO | <code>doupdate(3XC)</code> |

NAME reg_ci_callback – provide a component instrumentation with a transient program number

SYNOPSIS

```
cc [ flag ... ] file ... -ldmici [ library ... ]
#include <dmi/ci_callback_svc.hh>

u_long reg_ci_callback();
```

DESCRIPTION The **reg_ci_callback()** function provides a component instrumentation with a transient program number. The instrumentation uses this number to register its RPC service provider. The `prognum` member of the `DmiRegisterInfo` structure is populated with the return value of this function

RETURN VALUES Upon successful completion, the **reg_ci_callback()** function returns a transient program number of type `u_long`.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-level | Unafe |

SEE ALSO **attributes(5)**

| | |
|--------------------|--|
| NAME | regcmp, regex – compile and execute regular expression |
| SYNOPSIS | <pre>#include <libgen.h> char * regcmp(const char * <i>string1</i>, /* char * <i>string2</i> */ ..., int /*(char*)0*/); char * regex(const char * <i>re</i>, const char * <i>subject</i>, /* char * <i>ret0</i> */ ...); extern char * __loc1;</pre> |
| DESCRIPTION | <p>The regcmp() function compiles a regular expression (consisting of the concatenated arguments) and returns a pointer to the compiled form. The malloc(3C) function is used to create space for the compiled form. It is the user's responsibility to free unneeded space so allocated. A NULL return from regcmp() indicates an incorrect argument. regcmp(1) has been written to generally preclude the need for this routine at execution time.</p> <p>The regex() function executes a compiled pattern against the subject string. Additional arguments are passed to receive values back. The regex() function returns NULL on failure or a pointer to the next unmatched character on success. A global character pointer <code>__loc1</code> points to where the match began. The regcmp() and regex() functions were mostly borrowed from the editor ed(1); however, the syntax and semantics have been changed slightly. The following are the valid symbols and associated meanings.</p> <p>[] * . ^ This group of symbols retains its meaning as described on the regex(5) manual page.</p> <p>\$ Matches the end of the string; \ matches a newline.</p> <p>- Within brackets the minus means <i>through</i>. For example, [a-z] is equivalent to [abcd . . .xyz]. The - can appear as itself only if used as the first or last character. For example, the character class expression []- matches the characters] and -.</p> <p>+ A regular expression followed by + means <i>one or more times</i>. For example, [0-9]+ is equivalent to [0-9][0-9]*.</p> <p>{ <i>m</i> } { <i>m</i>, } { <i>m</i>, <i>u</i> } Integer values enclosed in { } indicate the number of times the preceding regular expression is to be applied. The value <i>m</i> is the minimum number and <i>u</i> is a number, less than 256, which is the maximum. If only <i>m</i> is present (that is, { <i>m</i> }), it indicates the exact number of times the regular expression is to be applied. The value { <i>m</i>, } is analogous to { <i>m</i>, <i>infinity</i> }. The plus (+) and star (*) operations are equivalent to { 1, } and { 0, } respectively.</p> |

- (...)\$ *n* The value of the enclosed regular expression is to be returned. The value will be stored in the (*n* + 1)th argument following the subject argument. At most, ten enclosed regular expressions are allowed. The **regex()** function makes its assignments unconditionally.
- (...) Parentheses are used for grouping. An operator, for example, *, +, { }, can work on a single character or a regular expression enclosed in parentheses. For example, (a*(cb+))*\$0 . By necessity, all the above defined symbols are special. They must, therefore, be escaped with a \\ (backslash) to be used as themselves.

EXAMPLES

EXAMPLE 1 Example matching a leading newline in the subject string.

The following example matches a leading newline in the subject string pointed at by cursor.

```
char *cursor, *newcursor, *ptr;
\011...
newcursor = regex((ptr = regcmp("^\
", (char *)0)), cursor);
free(ptr);
```

The following example matches through the string `Testing3` and returns the address of the character after the last matched character (the "4"). The string `Testing3` is copied to the character array `ret0`.

```
char ret0[9];
char *newcursor, *name;
\011...
name = regcmp("[A-Za-z][A-Za-z0-9]{0,7})$0", (char *)0);
newcursor = regex(name, "012Testing345", ret0);
```

The following example applies a precompiled regular expression in `file.i` (see **regcmp(1)**) against `string`.

```
#include "file.i"
char *string, *newcursor;
\011...
newcursor = regex(name, string);
```

FILES

/usr/ccs/lib/libgen.a

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

ed(1), **regcmp(1)**, **malloc(3C)**, **attributes(5)**, **regexp(5)**

NOTES

The user program may run out of memory if **regcmp()** is called iteratively without freeing the vectors no longer required.

When compiling multithreaded applications, the **_REENTRANT** flag must be defined on the compile line. This flag should only be used in multithreaded applications.

| | |
|--------------------|--|
| NAME | regcomp, regexec, regerror, regfree – regular expression matching |
| SYNOPSIS | <pre>#include <sys/types.h> #include <regex.h> int regcomp(regex_t * preg, const char * pattern, int cflags); int regexec(const regex_t * preg, const char * string, size_t nmatch, regmatch_t pmatch [], int eflags); size_t regerror(int errcode, const regex_t * preg, char * errbuf, size_t errbuf_size); void regfree(regex_t * preg);</pre> |
| DESCRIPTION | <p>These functions interpret <i>basic</i> and <i>extended</i> regular expressions (described on the regex(5) manual page).</p> <p>The structure type <code>regex_t</code> contains at least the following member:</p> <pre>size_t re_nsub Number of parenthesised subexpressions.</pre> <p>The structure type <code>regmatch_t</code> contains at least the following members:</p> <pre>regoff_t rm_so Byte offset from start of <i>string</i> to start of substring. regoff_t rm_eo Byte offset from start of <i>string</i> of the first character after the end of substring.</pre> |
| regcomp() | <p>The regcomp() function will compile the regular expression contained in the string pointed to by the <i>pattern</i> argument and place the results in the structure pointed to by <i>preg</i>. The <i>cflags</i> argument is the bitwise inclusive OR of zero or more of the following flags, which are defined in the header <code><regex.h></code>:</p> <pre>REG_EXTENDED Use Extended Regular Expressions. REG_ICASE Ignore case in match. REG_NOSUB Report only success/fail in regexec(). REG_NEWLINE Change the handling of NEWLINE characters, as described in the text.</pre> <p>The default regular expression type for <i>pattern</i> is a Basic Regular Expression. The application can specify Extended Regular Expressions using the <code>REG_EXTENDED</code> <i>cflags</i> flag.</p> <p>If the <code>REG_NOSUB</code> flag was not set in <i>cflags</i>, then regcomp() will set <i>re_nsub</i> to the number of parenthesised subexpressions (delimited by <code>\\(\\)</code> in basic regular expressions or <code>()</code> in extended regular expressions) found in <i>pattern</i>.</p> |

regex()

The **regex()** function compares the null-terminated string specified by *string* with the compiled regular expression *preg* initialized by a previous call to **regcomp()**. The *eflags* argument is the bitwise inclusive OR of zero or more of the following flags, which are defined in the header `<regex.h>`:

REG_NOTBOL The first character of the string pointed to by *string* is not the beginning of the line. Therefore, the circumflex character (`^`), when taken as a special character, will not match the beginning of *string*.

REG_NOTEOL The last character of the string pointed to by *string* is not the end of the line. Therefore, the dollar sign (`$`), when taken as a special character, will not match the end of *string*.

If *nmatch* is zero or **REG_NOSUB** was set in the *cflags* argument to **regcomp()**, then **regex()** will ignore the *pmatch* argument. Otherwise, the *pmatch* argument must point to an array with at least *nmatch* elements, and **regex()** will fill in the elements of that array with offsets of the substrings of *string* that correspond to the parenthesised subexpressions of *pattern*: *pmatch* [*i*]. *rm_so* will be the byte offset of the beginning and *pmatch* [*i*]. *rm_eo* will be one greater than the byte offset of the end of substring *i*. (Subexpression *i* begins at the *i* th matched open parenthesis, counting from 1.) Offsets in *pmatch* [0] identify the substring that corresponds to the entire regular expression. Unused elements of *pmatch* up to *pmatch* [*nmatch* - 1] will be filled with -1. If there are more than *nmatch* subexpressions in *pattern* (*pattern* itself counts as a subexpression), then **regex()** will still do the match, but will record only the first *nmatch* substrings.

When matching a basic or extended regular expression, any given parenthesised subexpression of *pattern* might participate in the match of several different substrings of *string*, or it might not match any substring even though the pattern as a whole did match. The following rules are used to determine which substrings to report in *pmatch* when matching regular expressions:

1. If subexpression *i* in a regular expression is not contained within another subexpression, and it participated in the match several times, then the byte offsets in *pmatch* [*i*] will delimit the last such match.
2. If subexpression *i* is not contained within another subexpression, and it did not participate in an otherwise successful match, the byte offsets in *pmatch* [*i*] will be -1. A subexpression does not participate in the match when:

* or `\\{\\}` appears immediately after the subexpression in a basic regular expression, or *, ?, or { } appears immediately after the subexpression in an extended regular expression, and the subexpression did not match (matched zero times)

or

| is used in an extended regular expression to select this subexpression or another, and the other subexpression matched.

3. If subexpression *i* is contained within another subexpression *j*, and *i* is not contained within any other subexpression that is contained within *j*, and a match of subexpression *j* is reported in *pmatch* [*j*], then the match or non-match of subexpression *i* reported in *pmatch* [*i*] will be as described in 1. and 2. above, but within the substring reported in *pmatch* [*j*] rather than the whole string.
4. If subexpression *i* is contained in subexpression *j*, and the byte offsets in *pmatch* [*j*] are -1, then the pointers in *pmatch* [*i*] also will be -1.
5. If subexpression *i* matched a zero-length string, then both byte offsets in *pmatch* [*i*] will be the byte offset of the character or NULL terminator immediately following the zero-length string.

If, when **regex()** is called, the locale is different from when the regular expression was compiled, the result is undefined.

If REG_NEWLINE is not set in *flags*, then a NEWLINE character in *pattern* or *string* will be treated as an ordinary character. If REG_NEWLINE is set, then newline will be treated as an ordinary character except as follows:

1. A NEWLINE character in *string* will not be matched by a period outside a bracket expression or by any form of a non-matching list.
2. A circumflex (^) in *pattern*, when used to specify expression anchoring will match the zero-length string immediately after a newline in *string*, regardless of the setting of REG_NOTBOL.
3. A dollar-sign (\$) in *pattern*, when used to specify expression anchoring, will match the zero-length string immediately before a newline in *string*, regardless of the setting of REG_NOTEOL.

regfree()

The **regfree()** function frees any memory allocated by **regcomp()** associated with *preg*.

The following constants are defined as error return values:

- | | |
|--------------|--|
| REG_NOMATCH | The regex() function failed to match. |
| REG_BADPAT | Invalid regular expression. |
| REG_ECOLLATE | Invalid collating element referenced. |
| REG_ETYPE | Invalid character class type referenced. |

| | |
|-------------|---|
| REG_EESCAPE | Trailing <code>\\</code> in pattern. |
| REG_ESUBREG | Number in <code>\\ digit</code> invalid or in error. |
| REG_EBRACK | <code>[]</code> imbalance. |
| REG_ENOSYS | The function is not supported. |
| REG_EPAREN | <code>\\(\\)</code> or <code>()</code> imbalance. |
| REG_EBRACE | <code>\\{ \\}</code> imbalance. |
| REG_BADBR | Content of <code>\\{ \\}</code> invalid: not a number, number too large, more than two numbers, first larger than second. |
| REG_ERANGE | Invalid endpoint in range expression. |
| REG_ESPACE | Out of memory. |
| REG_BADRPT | <code>?</code> , <code>*</code> or <code>+</code> not preceded by valid regular expression. |

regerror()

The **regerror()** function provides a mapping from error codes returned by **regcomp()** and **regex()** to unspecified printable strings. It generates a string corresponding to the value of the *errcode* argument, which must be the last non-zero value returned by **regcomp()** or **regex()** with the given value of *preg*. If *errcode* is not such a value, an error message indicating that the error code is invalid is returned.

If *preg* is a `NULL` pointer, but *errcode* is a value returned by a previous call to **regex()** or **regcomp()**, the **regerror()** still generates an error string corresponding to the value of *errcode*.

If the *errbuf_size* argument is not zero, **regerror()** will place the generated string into the buffer of size *errbuf_size* bytes pointed to by *errbuf*. If the string (including the terminating `NULL`) cannot fit in the buffer, **regerror()** will truncate the string and null-terminate the result.

If *errbuf_size* is zero, **regerror()** ignores the *errbuf* argument, and returns the size of the buffer needed to hold the generated string.

If the *preg* argument to **regex()** or **regfree()** is not a compiled regular expression returned by **regcomp()**, the result is undefined. A *preg* is no longer treated as a compiled regular expression after it is given to **regfree()**.

See **regex(5)** for BRE (Basic Regular Expression) Anchoring.

RETURN VALUES

On successful completion, the **regcomp()** function returns 0 . Otherwise, it returns an integer value indicating an error as described in <regex.h> , and the content of *preg* is undefined.

On successful completion, the **regexec()** function returns 0 . Otherwise it returns REG_NOMATCH to indicate no match, or REG_ENOSYS to indicate that the function is not supported.

Upon successful completion, the **regerror()** function returns the number of bytes needed to hold the entire generated string. Otherwise, it returns 0 to indicate that the function is not implemented.

The **regfree()** function returns no value.

ERRORS

No errors are defined.

USAGE

An application could use:

```
regerror(code, preg, (char *)NULL, (size_t)0)
```

to find out how big a buffer is needed for the generated string, `malloc` a buffer to hold the string, and then call **regerror()** again to get the string (see **malloc(3C)**). Alternately, it could allocate a fixed, static buffer that is big enough to hold most strings, and then use **malloc()** to allocate a larger buffer if it finds that this is too small.

EXAMPLES

EXAMPLE 1 Example to match string against the extended regular expression in pattern.

```
#include <regex.h>
/*
 * Match string against the extended regular expression in
 * pattern, treating errors as no match.
 *
 * return 1 for match, 0 for no match
 */

int
match(const char *string, char *pattern)
{
    int status;
    regex_t re;
    if (regcomp(&re, pattern, REG_EXTENDED | REG_NOSUB) != 0) {
        return(0);        /* report error */
    }
    status = regexec(&re, string, (size_t) 0, NULL, 0);
    regfree(&re);
    if (status != 0) {
        return(0);        /* report error */
    }
}
```

```

    }
    return(1);
}

```

The following demonstrates how the `REG_NOTBOL` flag could be used with `regexec()` to find all substrings in a line that match a pattern supplied by a user. (For simplicity of the example, very little error checking is done.)

```

(void) regcomp (&re, pattern, 0);
/* this call to regexec() finds the first match on the line */
error = regexec (&re, &buffer[0], 1, &pm, 0);
while (error == 0) {\011/* while matches found */
    /* substring found between pm.rm_so and pm.rm_eo */
    /* This call to regexec() finds the next match */
    error = regexec (&re, buffer + pm.rm_eo, 1, &pm, REG_NOTBOL);
}

```

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |
| CSI | Enabled |

SEE ALSO

`fnmatch(3C)`, `glob(3C)`, `malloc(3C)`, `setlocale(3C)`, `attributes(5)`, `regex(5)`

NOTES

The `regcomp()` function can be used safely in a multithreaded application as long as `setlocale(3C)` is not being called to change the locale.

| | |
|--------------------|--|
| NAME | regexpr, compile, step, advance – regular expression compile and match routines |
| SYNOPSIS | <pre>cc [flag...] [file...] -lgen [library...] #include <regexpr.h> char * compile(char * instring, char * expbuf, const char * endbuf); int step(const char * string, const char * expbuf); int advance(const char * string, const char * expbuf); extern char * loc1 , loc2 , locs ; extern int nbra , regerrno , reqlength ; extern char * braslist [], * braelist [];</pre> |
| DESCRIPTION | <p>These routines are used to compile regular expressions and match the compiled expressions against lines. The regular expressions compiled are in the form used by <code>ed(1)</code> .</p> <p>The parameter <i>instring</i> is a null-terminated string representing the regular expression.</p> <p>The parameter <i>expbuf</i> points to the place where the compiled regular expression is to be placed. If <i>expbuf</i> is <code>NULL</code> , <code>compile()</code> uses <code>malloc(3C)</code> to allocate the space for the compiled regular expression. If an error occurs, this space is freed. It is the user's responsibility to free unneeded space after the compiled regular expression is no longer needed.</p> <p>The parameter <i>endbuf</i> is one more than the highest address where the compiled regular expression may be placed. This argument is ignored if <i>expbuf</i> is <code>NULL</code> . If the compiled expression cannot fit in (<i>endbuf</i> – <i>expbuf</i>)bytes, <code>compile()</code> returns <code>NULL</code> and <code>regerrno</code> (see below) is set to 50.</p> <p>The parameter <i>string</i> is a pointer to a string of characters to be checked for a match. This string should be null-terminated.</p> <p>The parameter <i>expbuf</i> is the compiled regular expression obtained by a call of the function <code>compile()</code> .</p> <p>The function <code>step()</code> returns non-zero if the given string matches the regular expression, and zero if the expressions do not match. If there is a match, two external character pointers are set as a side effect to the call to <code>step()</code> . The variables set in <code>step()</code> are <code>loc1</code> and <code>loc2</code> . <code>loc1</code> is a pointer to the first character that matched the regular expression. The variable <code>loc2</code> points to the character after the last character that matches the regular expression. Thus if</p> |

the regular expression matches the entire line, `loc1` points to the first character of *string* and `loc2` points to the null at the end of *string*.

The purpose of **step()** is to step through the *string* argument until a match is found or until the end of *string* is reached. If the regular expression begins with `^`, **step()** tries to match the regular expression at the beginning of the string only.

The **advance()** function is similar to **step()**; but, it only sets the variable `loc2` and always restricts matches to the beginning of the string.

If one is looking for successive matches in the same string of characters, `locs` should be set equal to `loc2`, and **step()** should be called with *string* equal to `loc2`. `locs` is used by commands like `ed` and `sed` so that global substitutions like `s/y*/g` do not loop forever, and is `NULL` by default.

The external variable `nbra` is used to determine the number of subexpressions in the compiled regular expression. `braslist` and `braelist` are arrays of character pointers that point to the start and end of the `nbra` subexpressions in the matched string. For example, after calling **step()** or **advance()** with *string* `sabcdefg` and regular expression `\\(abcdef\\)`, `braslist[0]` will point at `a` and `braelist[0]` will point at `g`. These arrays are used by commands like `ed` and `sed` for substitute replacement patterns that contain the `\\ n` notation for subexpressions.

Note that it is not necessary to use the external variables `regerrno`, `nbra`, `loc1`, `loc2`, `locs`, `braelist`, and `braslist` if one is only checking whether or not a string matches a regular expression.

EXAMPLES

EXAMPLE 1 The following is similar to the regular expression code from `grep`:

```
#include<regexpr.h>
. . .
if(compile(*argv, (char *)0, (char *)0) == (char *)0)
    \011regerr(regerrno);
. . .
if (step(linebuf, expbuf) \011
    succeed());
```

RETURN VALUES

If **compile()** succeeds, it returns a non- `NULL` pointer whose value depends on *expbuf*. If *expbuf* is non- `NULL`, **compile()** returns a pointer to the byte after the last byte in the compiled regular expression. The length of the compiled regular expression is stored in `reglength`. Otherwise, **compile()** returns a pointer to the space allocated by `malloc(3C)`.

The functions **step()** and **advance()** return non-zero if the given string matches the regular expression, and zero if the expressions do not match.

ERRORS

If an error is detected when compiling the regular expression, a `NULL` pointer is returned from `compile()` and `regerrno` is set to one of the non-zero error numbers indicated below:

| ERROR | MEANING |
|-------|---------------------------------------|
| 11 | Range endpoint too large. |
| 16 | Bad Number. |
| 25 | "\digit" out of range. |
| 36 | Illegal or missing delimiter. |
| 41 | No remembered string search. |
| 42 | \(-\) imbalance. |
| 43 | Too many \(\. |
| 44 | More than 2 numbers given in \[-\]. |
| 45 | } expected after \. |
| 46 | First number exceeds second in \{-\}. |
| 49 | [] imbalance. |
| 50 | Regular expression overflow. |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`ed(1)`, `grep(1)`, `sed(1)`, `malloc(3C)`, `attributes(5)`, `regexp(5)`

NOTES

When compiling multi-threaded applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multi-threaded applications.

| NAME | remainder – remainder function | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | #include <math.h> double remainder(double x, double y); | | | | |
| DESCRIPTION | The remainder() function returns the floating point remainder $r = x - ny$ when y is non-zero. The value n is the integral value nearest the exact value x/y . When $ n - x/y = \frac{1}{2}$, the value n is chosen to be even. The behaviour of remainder() is independent of the rounding mode. | | | | |
| RETURN VALUES | The remainder() function returns the floating point remainder $r = x - ny$ when y is non-zero. When y is 0, remainder() returns NaN. and sets <code>errno</code> to EDOM. If the value of x is $\pm\text{Inf}$, remainder() returns NaN and sets <code>errno</code> to EDOM. If x or y is NaN, then the function returns NaN. | | | | |
| ERRORS | The remainder() function will fail if: EDOM The y argument is 0 or the x argument is positive or negative infinity. | | | | |
| USAGE | The remainder() function computes the remainder $x \text{ REM } y$ required by ANSI/IEEE 754 (IEC 559). | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | fmod(3M) , attributes(5) | | | | |

| NAME | remove – remove file | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | #include <stdio.h> int remove (const char *path); | | | | |
| DESCRIPTION | The remove() function causes the file or empty directory whose name is the string pointed to by <i>path</i> to be no longer accessible by that name. A subsequent attempt to open that file using that name will fail, unless the file is created anew. For files, remove() is identical to unlink() . For directories, remove() is identical to rmdir() . See rmdir(2) and unlink(2) for a detailed list of failure conditions. | | | | |
| RETURN VALUES | Upon successful completion, remove() returns 0. Otherwise, it returns -1 and sets <code>errno</code> to indicate an error. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | rmdir(2) , unlink(2) , attributes(5) | | | | |

| | |
|----------------------|--|
| NAME | resetty, savetty – restore/save terminal modes |
| SYNOPSIS | <pre>#include <curses.h> int resetty(void); int savetty(void);</pre> |
| DESCRIPTION | The savetty() and resetty() functions save and restore the terminal state, respectively. The savetty() function saves the current state in a buffer; the resetty() function restores the state to that stored in the buffer at the time of the last savetty() call. |
| RETURN VALUES | On success, these functions return <code>OK</code> . Otherwise, they return <code>ERR</code> . |
| ERRORS | None. |

| | |
|--------------------|--|
| NAME | resolver, res_init, res_mkquery, res_mkupdate, res_mkupdrec, res_query, res_search, res_send, res_update, dn_comp, dn_expand – resolver routines |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lresolv -lsocket -lnsl [<i>library</i> ...] #include <sys/types.h> #include <netinet/in.h> #include <arpa/nameser.h> #include <resolv.h> int res_init(void); int res_mkquery(int op, const char * dname, int class, int type, const char * data, int datalen, struct rrec * newrr, uchar_t * buf, int buflen); int res_mkupdate(ns_updrec ** rrecp_in, uchar * buf, int length); ns_updrec * res_mkupdrec(int section, const char * dname, uint_t class, uint_t type, uint_t ttl); int res_query(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_search(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_send(uchar_t * msg, int msglen, uchar_t * answer, int anslen); int res_update(ns_updrec * rrecp_in); int dn_comp(const char * exp_dn, uchar_t * comp_dn, int length, uchar_t ** dnptrs, uchar_t ** lastdnptr); int dn_expand(const uchar_t * msg, const uchar_t * eomorig, uchar_t * comp_dn, char exp_dn, int length); </pre> |
| DESCRIPTION | These routines are used for making, sending, and interpreting query and reply messages passed to and from Internet domain name servers. The res_update() |

and **res_mkupdrec()** routines are used to dynamically update the name server with resource records.

The global structure `_res` holds options and state information. Option values can be set to affect the collective behavior of groups of resolver library routines. However, most resolver library routines use reasonable defaults so that the explicit enabling of an option is rarely required.

The library manual page entry for the resolver library includes public domain routines beyond those described here. See `libresolv(4)`. Those function names that are exported but are not explained here are lower-level routines called by these routines. Their direct use is discouraged. If you do make direct use of unsupported routines, you do so at considerable added risk and with no expectation of documentation or other support beyond that available publicly.

Options for the resolver library are stored as a single bit mask containing the bitwise-OR sum of the options enabled. The options stored in `_res.options` are those defined in `<resolv.h>` and as follows. The field `_res.options` is a member of the `_res` structure.

| | |
|---------------------------|--|
| <code>RES_INIT</code> | True if the initial name server address and default domain name are initialized, that is, res_init() has been called. |
| <code>RES_DEBUG</code> | Print debugging messages. |
| <code>RES_AAONLY</code> | Accept authoritative answers only. With this option, res_send() will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented. |
| <code>RES_USEVC</code> | Use TCP connections for queries instead of UDP datagrams. |
| <code>RES_PRIMARY</code> | Query primary server only. This option is not implemented. |
| <code>RES_IGNTC</code> | Unused currently. Ignore truncation errors; that is, do not retry with TC P. |
| <code>RES_RECURSE</code> | Set the recursion-desired bit in queries. This is the default. res_send() does not do iterative queries and expects the name server to handle recursion. |
| <code>RES_DEFNAMES</code> | If set, res_search() appends the default domain name to single-component names (names that do not contain a dot). This is useful only in |

| | | |
|------------|---------------|--|
| | | programs that regularly do many queries. UDP should be the normal mode used. |
| | RES_DNSRCH | Enables searching up through the current domain tree. If this option is set, res_search() searches for host names in the current domain and in parent domains. This is used by the standard host lookup routine gethostbyname(3N) . This option is enabled by default. |
| | RES_NOALIASES | This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option. |
| res_init | | <p>If the system initialization file <code>resolv.conf</code> exists, res_init() reads it to get the default domain name, the search list, and the Internet address of the local name server or servers. See resolv.conf(4) . If no server is configured by the local <code>resolv.conf</code> file, <code>res_init</code> tries to obtain name resolution services from the host on which it is running.</p> <p>The res_init() function also sets the <code>RES_INIT</code> field of the <code>_res</code> global structure so that other service routines (res_search()) can determine for certain whether it needs to be called first before other processing begins.</p> <p>In the absence of a <code>resolv.conf</code> configuration file, the current domain is either set to the value of the environmental variable <code>LOCALDOMAIN</code>, derived from the domain name or derived from the host name. See domainname(1M) . The current domain name as defined in the system initialization file <code>resolv.conf</code> can be overridden by the environment variable <code>LOCALDOMAIN</code> . This environment variable may contain several blank-separated tokens if you wish to override the <i>search list</i> on a per-process basis. This is similar to the search command in the configuration file. Another environment variable (<code>RES_OPTIONS</code>) can be set to override certain internal resolver options. Otherwise, these options are set by changing fields in the global <code>_res</code> structure or they are inherited from the configuration file's <i>options</i> command. The syntax of the <code>RES_OPTIONS</code> environment variable is explained in resolv.conf(4) .</p> <p>The initializations performed by res_init() can be invoked explicitly with this function. However, they are normally performed automatically as a result of a call to one of the other resolver routines.</p> |
| res_search | | res_search() formulates and sends a normal query (<code>QUERY</code>) message, and stores the response in a buffer supplied by the caller. |

The parameters *class* and *type* define the class and type of query. See `<arpa/nameser.h>`. The response is returned in the user-supplied buffer *answer*. `res_search` returns the length of *answer* in *anslen*. Like the other resolver routines, `res_search()` calls `res_init()` if the `RES_INIT` flag is not enabled.

The `res_search()` function acts in a similar manner as `res_query()`, except that it can implement the default and search rules controlled by the `RES_DEFNAMES` and `RES_DNSRCH` options.

The parameter *dname* is the domain name. However, if *dname* consists of a single-component name and the `RES_DEFNAMES` flag is enabled (the default), *dname* is appended with the current domain name.

If the `RES_DNSRCH` flag is enabled, `res_search()` searches up the current domain tree until an answer has been retrieved or an unrecoverable error has been encountered. `res_search()` returns the first successful reply.

`res_mkquery`

The `res_mkquery()` function constructs a standard query message and places it in *buf*.

`res_mkquery()` returns the size of the query or `-1` if the query is larger than *buflen*. The *op* parameter is usually `QUERY`, but can be any of the query types defined in `<arpa/nameser.h>`.

The parameter *dname* is the domain name for the query.

The parameters *class* and *type* define the class and type of query (see `<arpa/nameser.h>`). The parameter *data* is the resource record; *datalen* is the length of the record.

newrr is unused and should be specified as a null pointer (`0`).

`res_query`

The `res_query` function provides an interface to most of the server query mechanism. It constructs a query, sends it to the local server, awaits a response, and makes preliminary checks on the reply. The query requests information of the specified *type* and *class* for the specified fully-qualified domain name *dname*. The reply message is left in the *answer* buffer with length *anslen* supplied by the caller. The `res_mkquery` and `res_send` routines described elsewhere in this section are lower-level routines that `res_query` calls.

`res_send`

`res_send()` sends a pre-formatted query to name servers and returns an answer. It calls `res_init()` if `RES_INIT` is not set, send the query to the local name server, and handle timeouts and retries. *msg* is the query sent; *msglen* is its length. *answer* is the response returned. The length of the response is stored in *anslen*. `res_send()` returns the length of the response or `-1` if there were errors.

| | |
|--------------|--|
| dn_expand | <p>dn_expand() expands the compressed domain name given by the pointer <i>comp_dn</i> into a full domain name. Expanded names are converted to upper case. The compressed name is contained in a query or reply message; <i>msg</i> is a pointer to the beginning of that message. Expanded names are stored in the buffer referenced by the <i>exp_dn</i> buffer of size <i>length</i>, which should be large enough to hold the expanded result.</p> <p>dn_expand() returns the size of the compressed name, or -1 if there was an error.</p> |
| dn_comp | <p>dn_comp() compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. dn_comp() returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>. <i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with NULL. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling dn_comp() is to update the list of pointers for labels inserted into the message by dn_comp() as the name is compressed. However, if <i>lastdnptr</i> is NULL, dn_comp() does not update the list of labels. If <i>dnptrs</i> is NULL, names are not compressed.</p> |
| res_mkupdrec | <p>res_mkupdrec() takes the elements of a resource record as its arguments, for instance, <i>section</i>, <i>domainname</i>, <i>class</i>, <i>type</i>, and <i>tll</i>, and returns a pointer to a linked list <i>ns_updrec</i>. It returns NULL upon failure.</p> |
| res_update | <p>res_update() takes a linked list of resource records <i>ns_updrec</i> as its only argument and separates the records into groups such that all records in a group will belong to a single name server. It creates a dynamic update packet for each zone and sends it to the name server and awaits an answer. Upon success, res_update() returns the number of zones updated. It returns <0 upon error.</p> |
| res_mkupdate | <p>res_mkupdate() creates update packets by running through the elements of the <i>ns_updrec</i> link list. It is called by res_update() to create packets for res_send() to send to the name server. res_mkupdate() returns the actual size of the packet, or</p> <ul style="list-style-type: none"> -1 Error in reading a word or number in the <i>rdata</i> portion for update packets. -2 The length of the packet passed is insufficient. -3 The zone section is not the first section in the linked list, or the section order has a problem. -4 A number overflow. |

-5 Unknown operations or no records.
 If an error occurs it could leave the zones being updated in an inconsistent state. For example, a record might be successfully added to the forward lookup zone but the corresponding PTR record might have failed to update in the reverse lookup tables.

FILES

/etc/resolv.conf

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

domainname(1M), **in.named(1M)**, **nstest(1M)**, **gethostbyname(3N)**, **libresolv(4)**, **resolv.conf(4)**, **attributes(5)**

Lottor, M., *Domain Administrators Operators Guide*, RFC 1033, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide*, RFC 1032, SRI International, Menlo Park, Calif., November 1987.

Vixie, Paul;Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND* (public domain), Internet Software Consortium, 1996.

NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

| NAME | rewind – reset file position indicator in a stream | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | #include <stdio.h> void rewind (FILE * <i>stream</i>); | | | | |
| DESCRIPTION | The call: rewind(<i>stream</i>) is equivalent to: (void) fseek (<i>stream</i> , 0L, SEEK_SET) except that rewind() also clears the error indicator. | | | | |
| RETURN VALUES | The rewind() function returns no value. | | | | |
| ERRORS | Refer to fseek(3S) with the exception of EINVAL which does not apply. | | | | |
| USAGE | Because rewind() does not return a value, an application wishing to detect errors should clear <code>errno</code> , then call rewind() , and if <code>errno</code> is non-zero, assume an error has occurred. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | fseek(3S) , attributes(5) | | | | |

| NAME | rewinddir – reset position of directory stream to the beginning of a directory | | | | |
|----------------------|--|----------------|-----------------|----------|------|
| SYNOPSIS | <pre>#include <sys/types.h> #include <dirent.h> void rewinddir(DIR *dirp);</pre> | | | | |
| DESCRIPTION | <p>The rewinddir() function resets the position of the directory stream to which <i>dirp</i> refers to the beginning of the directory. It also causes the directory stream to refer to the current state of the corresponding directory, as a call to opendir(3C) would have done. If <i>dirp</i> does not refer to a directory stream, the effect is undefined.</p> <p>After a call to the fork(2) function, either the parent or child (but not both) may continue processing the directory stream using readdir(3C), rewinddir() or seekdir(3C). If both the parent and child processes use these functions, the result is undefined.</p> | | | | |
| RETURN VALUES | The rewinddir() function does not return a value. | | | | |
| ERRORS | No errors are defined. | | | | |
| USAGE | The rewinddir() function should be used in conjunction with opendir() , readdir() , and closedir(3C) to examine the contents of the directory. This method is recommended for portability. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe | | | | |
| SEE ALSO | fork(2) , closedir(3C) , opendir(3C) , readdir(3C) , seekdir(3C) , attributes(5) | | | | |

NAME rexec - return stream to a remote command

SYNOPSIS `cc [flag ...] file ... -lsocket -lnsl [library ...]`

```
int rexec(char **ahost, unsigned short inport, const char *user, const char *passwd, const char *cmd, int *fd2p);
```

DESCRIPTION **rexec()** looks up the host **ahost* using **gethostbyname(3N)**, returning `-1` if the host does not exist. Otherwise **ahost* is set to the standard name of the host. If a username and password are both specified, then these are used to authenticate to the foreign host; otherwise the user's `.netrc` file in his home directory is searched for appropriate information. If all this fails, the user is prompted for the information.

The port *inport* specifies which well-known DARPA Internet port to use for the connection. The protocol for connection is described in detail in **in.rexecd(1M)**.

If the call succeeds, a socket of type `SOCK_STREAM` is returned to the caller, and given to the remote command as its standard input and standard output. If *fd2p* is non-zero, then an auxiliary channel to a control process will be setup, and a file descriptor for it will be placed in **fd2p*. The control process will return diagnostic output (file descriptor 2, the standard error) from the command on this channel, and will also accept bytes on this channel as signal numbers, to be forwarded to the process group of the command. If *fd2p* is 0, then the standard error (file descriptor 2 of the remote command) will be made the same as its standard output and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.

RETURN VALUES If **rexec()** succeeds, a file descriptor number, which is a socket of type `SOCK_STREAM`, is returned by the routine. **ahost* is set to the standard name of the host, and if *fd2p* is not `NULL`, a file descriptor number is placed in **fd2p* which represents the command's standard error stream.

If **rexec()** fails, `-1` is returned.

ATTRIBUTES See **attributes (5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO **in.rexecd(1M)**, **gethostbyname(3N)**, **getservbyname(3N)**, **socket(3N)**, **attributes(5)**

NOTES

There is no way to specify options to the **socket()** call that **rexec()** makes.

This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

| NAME | rint – round-to-nearest integral value | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | cc [<i>flag ...</i>] <i>file ...</i> -lm [<i>library ...</i>] #include <math.h> double rint(double <i>x</i>); | | | | |
| DESCRIPTION | The rint() function returns the integral value (represented as a double) nearest <i>x</i> in the direction of the current IEEE754 rounding mode. If the current rounding mode rounds toward negative infinity, then rint() is identical to floor(3M) . If the current rounding mode rounds toward positive infinity, then rint() is identical to ceil(3M) . | | | | |
| RETURN VALUES | Upon successful completion, the rint() function returns the integer (represented as a double precision number) nearest <i>x</i> in the direction of the current IEEE754 rounding mode. When <i>x</i> is ±Inf, rint() returns <i>x</i> . If the value of <i>x</i> is NaN, NaN is returned. | | | | |
| ERRORS | No errors will occur. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | ceil(3M) , floor(3M) , isnan(3M) , attributes(5) | | | | |

| | |
|----------------------|--|
| NAME | riponline - reserve screen line for dedicated purpose |
| SYNOPSIS | #include <curses.h> int riponline (int <i>line</i> , int (* <i>init</i>)(WINDOW * <i>win</i> , int <i>width</i>); |
| PARAMETERS | <p><i>line</i> determines whether the screen line being reserved comes from the top of <code>stdscr</code> (<i>line</i> is positive) or the bottom (<i>line</i> is negative).</p> <p><i>init</i> Is a pointer to a function that initializes the one-line window.</p> <p><i>win</i> Is a pointer to one-line window created by this function.</p> <p><i>width</i> Is the number of columns in the window pointed to by the <i>win</i> parameter.</p> |
| DESCRIPTION | <p>The riponline() function reserves a screen line as a one line window.</p> <p>To use this function, it must be called before you call initscr(3XC) or newterm(3XC). When initscr() or newterm() is called, so is the function pointed to by <i>init</i>. The function pointed to by <i>init</i> takes two arguments: a pointer to the one-line window and the number of columns in that window. This function cannot use the <code>LINES</code> or <code>COLS</code> variables and cannot call wrefresh(3XC) or doupdate(3XC), but may call wnoutrefresh(3XC).</p> |
| RETURN VALUES | The riponline() function always returns <code>OK</code> . |
| ERRORS | None. |
| SEE ALSO | doupdate(3XC) , initscr(3XC) , slk_attroff(3XC) |

| | | | |
|-----------------------------------|--|----------------------|---|
| NAME | rpc - library routines for remote procedure calls | | |
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lnsl [<i>library ...</i>] #include <rpc/rpc.h> #include <netconfig.h></pre> | | |
| DESCRIPTION | <p>These routines allow C language programs to make procedure calls on other machines across a network. First, the client sends a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply.</p> <p>All RPC routines require the header <code><rpc/rpc.h></code>. Routines that take a <code>netconfig</code> structure also require that <code><netconfig.h></code> be included. Applications using RPC and XDR routines should be linked with the <code>libnsl</code> library.</p> | | |
| Multithread Considerations | <p>In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). Defining this flag enables a thread-specific version of <code>rpc_createerr</code>. See <code>rpc_clnt_create(3N)</code>.</p> <p>When used in multithreaded applications, client-side routines are MT-Safe. CLIENT handles can be shared between threads; however, in this implementation, requests by different threads are serialized (that is, the first request will receive its results before the second request is sent). See <code>rpc_clnt_create(3N)</code>.</p> <p>When used in multithreaded applications, server-side routines are usually Unsafe. In this implementation the service transport handle, <code>SVCXPRT</code> contains a single data area for decoding arguments and encoding results. See <code>rpc_svc_create(3N)</code>. Therefore, this structure cannot be freely shared between threads that call functions that do this. Routines that are affected by this restriction are marked as unsafe for MT applications. See <code>rpc_svc_calls(3N)</code>.</p> | | |
| Nettyp | <p>Some of the high-level RPC interface routines take a <i>nettype</i> string as one of the parameters (for example, <code>clnt_create()</code>, <code>svc_create()</code>, <code>rpc_reg()</code>, <code>rpc_call()</code>). This string defines a class of transports which can be used for a particular application.</p> <p><i>nettype</i> can be one of the following:</p> <table border="0" style="margin-left: 20px;"> <tr> <td style="padding-right: 20px;"><code>netpath</code></td> <td>Choose from the transports which have been indicated by their token names in the NETPATH environment variable. If NETPATH is unset or NULL, it defaults to <code>visible</code>. <code>netpath</code> is the default <i>nettype</i>.</td> </tr> </table> | <code>netpath</code> | Choose from the transports which have been indicated by their token names in the NETPATH environment variable. If NETPATH is unset or NULL, it defaults to <code>visible</code> . <code>netpath</code> is the default <i>nettype</i> . |
| <code>netpath</code> | Choose from the transports which have been indicated by their token names in the NETPATH environment variable. If NETPATH is unset or NULL, it defaults to <code>visible</code> . <code>netpath</code> is the default <i>nettype</i> . | | |

`visible` Choose the transports which have the visible flag (`v`) set in the `/etc/netconfig` file.

`circuit_v` This is same as `visible` except that it chooses only the connection oriented transports (semantics `tpi_cots` or `tpi_cots_ord`) from the entries in the `/etc/netconfig` file.

`datagram_v` This is same as `visible` except that it chooses only the connectionless datagram transports (semantics `tpi_clts`) from the entries in the `/etc/netconfig` file.

`circuit_n` This is same as `netpath` except that it chooses only the connection oriented datagram transports (semantics `tpi_cots` or `tpi_cots_ord`).

`datagram_n` This is same as `netpath` except that it chooses only the connectionless datagram transports (semantics `tpi_clts`).

`udp` This refers to Internet UDP.

`tcp` This refers to Internet TCP.

If `nettype` is NULL, it defaults to `netpath`. The transports are tried in left to right order in the `NETPATH` variable or in top to down order in the `/etc/netconfig` file.

Derived Types

In a 64-bit environment, the derived types are defined as follows:

| | | |
|----------------------|-----------------------|----------------------------|
| <code>typedef</code> | <code>uint32_t</code> | <code>rpcprog_t;</code> |
| <code>typedef</code> | <code>uint32_t</code> | <code>rpcvers_t;</code> |
| <code>typedef</code> | <code>uint32_t</code> | <code>rpcproc_t;</code> |
| <code>typedef</code> | <code>uint32_t</code> | <code>rpcprot_t;</code> |
| <code>typedef</code> | <code>uint32_t</code> | <code>rpcport_t;</code> |
| <code>typedef</code> | <code>int32_t</code> | <code>rpc_inline_t;</code> |

In a 32-bit environment, the derived types are defined as follows:

| | | |
|----------------------|----------------------------|-------------------------|
| <code>typedef</code> | <code>unsigned long</code> | <code>rpcprog_t;</code> |
| <code>typedef</code> | <code>unsigned long</code> | <code>rpcvers_t;</code> |
| <code>typedef</code> | <code>unsigned long</code> | <code>rpcproc_t;</code> |
| <code>typedef</code> | <code>unsigned long</code> | <code>rpcprot_t;</code> |

Data Structures

Some of the data structures used by the RPC package are shown below.

The AUTH Structure

```

typedef          unsigned long      rpcport_t;
typedef          long               rpc_inline_t;

union des_block {
    struct {
        u_int32  high;
        u_int32  low;
    } key;
    char  c[8];
};
typedef union des_block des_block;
extern bool_t xdr_des_block();
/*
 * Authentication info. Opaque to client.
 */
struct opaque_auth {
    enum_t oa_flavor;          /* flavor of auth */
    caddr_t oa_base;          /* address of more auth stuff */
    uint_t oa_length;         /* not to exceed MAX_AUTH_BYTES */
};
/*
 * Auth handle, interface to client side authenticators.
 */
typedef struct {
    struct opaque_auth ah_cred;
    struct opaque_auth ah_verf;
    union des_block ah_key;
    struct auth_ops {
        void(*ah_nextverf)();
        int(*ah_marshall)();      /* nextverf & serialize */
        int(*ah_validate)();     /* validate verifier */
        int(*ah_refresh)();      /* refresh credentials */
        void(*ah_destroy)();     /* destroy this structure */
    } *ah_ops;
    caddr_t ah_private;
} AUTH;

```

The CLIENT Structure

```

/*
 * Client rpc handle.
 * Created by individual implementations.
 * Client is responsible for initializing auth.
 */
typedef struct {
    AUTH *cl_auth;          /* authenticator */
    struct clnt_ops {
        enum clnt_stat (*cl_call)();      /* call remote procedure */
        void (*cl_abort)();             /* abort a call */
    }

```

**The SVCXPRT
Structure**

```

        void (*cl_geterr)();           /* get specific error code */
        bool_t (*cl_freeres)();       /* frees results */
        void (*cl_destroy)();         /* destroy this structure */
        bool_t (*cl_control)();       /* the ioctl() of rpc */
        int (*cl_settimers)();        /* set rpc level timers */
    } *cl_ops;
    caddr_t cl_private;               /* private stuff */
    char *cl_netid;                   /* network identifier */
    char *cl_tp;                       /* device name */
} CLIENT;

```

```

enum xpirt_stat {
    XPRT_DIED,
    XPRT_MOREREQS,
    XPRT_IDLE
};
/*
 * Server side transport handle
 */
typedef struct {
    int xp_fd;                        /* file descriptor for the
                                     /* obsolete */
    ushort_t xp_port;
    struct xp_ops {
        bool_t (*xp_rcv)(); /* receive incoming requests */
        enum xpirt_stat (*xp_stat)(); /* get transport status */
        bool_t (*xp_getargs)(); /* get arguments */
        bool_t (*xp_reply)(); /* send reply */
        bool_t (*xp_freeargs)(); /* free mem allocated
                                     for args */
        void (*xp_destroy)(); /* destroy this struct */
    } *xp_ops;
    int xp_addrlen;                  /* length of remote addr.
                                     Obsolete */
    char *xp_tp;                     /* transport provider device
                                     name */
    char *xp_netid;                  /* network identifier */
    struct netbuf xp_ltaddr;          /* local transport address */
    struct netbuf xp_rtaddr;          /* remote transport address */
    char xp_raddr[16];               /* remote address. Obsolete */
    struct opaque_auth xp_verf;      /* raw response verifier */
    caddr_t xp_p1;                   /* private: for use
                                     by svc ops */
    caddr_t xp_p2;                   /* private: for use
                                     by svc ops */
    caddr_t xp_p3;                   /* private: for use
                                     by svc lib */
    int xp_type                       /* transport type */
} SVCXPRT;

```

**The svc_req
Structure**

```

struct svc_req {
    rpcprog_t rq_prog;               /* service program number */

```

The XDR Structure

```

    rpcvers_t rq_vers;           /* service protocol version */
    rpcproc_t rq_proc;          /* the desired procedure */
    struct opaque_auth rq_cred; /* raw creds from the wire */
    caddr_t rq_clntcred;        /* read only cooked cred */
    SVCXPRT *rq_xprt;           /* associated transport */
};

/*
 * XDR operations.
 * XDR_ENCODE causes the type to be encoded into the stream.
 * XDR_DECODE causes the type to be extracted from the stream.
 * XDR_FREE can be used to release the space allocated by an XDR_DECODE
 * request.
 */
enum xdr_op {
    XDR_ENCODE=0,
    XDR_DECODE=1,
    XDR_FREE=2
};
/*
 * This is the number of bytes per unit of external data.
 */
#define BYTES_PER_XDR_UNIT (4)
#define RNDUP(x) (((x) + BYTES_PER_XDR_UNIT - 1) /
    BYTES_PER_XDR_UNIT) \ * BYTES_PER_XDR_UNIT
/*
 * A xdrproc_t exists for each data type which is to be encoded or
 * decoded. The second argument to the xdrproc_t is a pointer to
 * an opaque pointer. The opaque pointer generally points to a
 * structure of the data type to be decoded. If this points to 0,
 * then the type routines should allocate dynamic storage of the
 * appropriate size and return it.
 * bool_t (*xdrproc_t)(XDR *, caddr_t *);
 */
typedef bool_t (*xdrproc_t)();
/*
 * The XDR handle.
 * Contains operation which is being applied to the stream,
 * an operations vector for the particular implementation
 */
typedef struct {
    enum xdr_op x_op; /* operation; fast additional param */
    struct xdr_ops {
        bool_t (*x_getlong)(); /* get long from underlying stream */
        bool_t (*x_putlong)(); /* put long to underlying stream */
        bool_t (*x_getbytes)(); /* get bytes from underlying stream */
        bool_t (*x_putbytes)(); /* put bytes to underlying stream */
        uint_t (*x_getpostn)(); /* returns bytes off from beginning */
        bool_t (*x_setpostn)(); /* reposition the stream */
        rpc_inline_t (*x_inline)(); /* buf quick ptr to buffered data */
        void (*x_destroy)(); /* free privates of this xdr_stream */
        bool_t (*x_control)(); /* changed/retrieve client object info*/
    };
};

```

```

bool_t      (*x_getint32)();      /* get int from underlying stream */
bool_t      (*x_putint32)();     /* put int to underlying stream */

} *x_ops;

caddr_t     x_public;            /* users' data */
caddr_t     x_priv              /* pointer to private data */
caddr_t     x_base;             /* private used for position info */
int         x_handy;            /* extra private word */
XDR;

```

Index to Routines

The following table lists RPC routines and the manual reference pages on which they are described:

| RPC Routine | Manual Reference Page |
|-------------------------|----------------------------|
| auth_destroy | rpc_clnt_auth(3N) |
| authdes_create | rpc_soc(3N) |
| authdes_getucred | secure_rpc(3N) |
| authdes_seccreate | secure_rpc(3N) |
| authkerb_getucred | kerberos_rpc(3N) |
| authkerb_seccreate | kerberos_rpc(3N) |
| authnone_create | rpc_clnt_auth(3N) |
| authsys_create | rpc_clnt_auth(3N) |
| authsys_create_default | rpc_clnt_auth(3N) |
| authunix_create | rpc_soc(3N) |
| authunix_create_default | rpc_soc(3N) |
| callrpc | rpc_soc(3N) |
| clnt_broadcast | rpc_soc(3N) |
| clnt_call | rpc_clnt_calls(3N) |
| clnt_control | rpc_clnt_create(3N) |
| clnt_create | rpc_clnt_create(3N) |
| clnt_destroy | rpc_clnt_create(3N) |

| | |
|---------------------|---------------------|
| clnt_dg_create | rpc_clnt_create(3N) |
| clnt_freeres | rpc_clnt_calls(3N) |
| clnt_geterr | rpc_clnt_calls(3N) |
| clnt_pcreateerror | rpc_clnt_create(3N) |
| clnt_perrno | rpc_clnt_calls(3N) |
| clnt_perror | rpc_clnt_calls(3N) |
| clnt_raw_create | rpc_clnt_create(3N) |
| clnt_spccreateerror | rpc_clnt_create(3N) |
| clnt_sperrno | rpc_clnt_calls(3N) |
| clnt_spperror | rpc_clnt_calls(3N) |
| clnt_tli_create | rpc_clnt_create(3N) |
| clnt_tp_create | rpc_clnt_create(3N) |
| clnt_udpcreate | rpc_soc(3N) |
| clnt_vc_create | rpc_clnt_create(3N) |
| clntraw_create | rpc_soc(3N) |
| clnttcp_create | rpc_soc(3N) |
| clntudp_bufcreate | rpc_soc(3N) |
| get_myaddress | rpc_soc(3N) |
| getnetname | secure_rpc(3N) |
| host2netname | secure_rpc(3N) |
| key_decryptsession | secure_rpc(3N) |
| key_encryptsession | secure_rpc(3N) |
| key_gendes | secure_rpc(3N) |
| key_setsecret | secure_rpc(3N) |

| | |
|--------------------|---------------------------|
| netname2host | secure_rpc(3N) |
| netname2user | secure_rpc(3N) |
| pmap_getmaps | rpc_soc(3N) |
| pmap_getport | rpc_soc(3N) |
| pmap_rmtcall | rpc_soc(3N) |
| pmap_set | rpc_soc(3N) |
| pmap_unset | rpc_soc(3N) |
| rac_drop | rpc_rac(3N) |
| rac_poll | rpc_rac(3N) |
| rac_recv | rpc_rac(3N) |
| rac_send | rpc_rac(3N) |
| registerrpc | rpc_soc(3N) |
| rpc_broadcast | rpc_clnt_calls(3N) |
| rpc_broadcast_exp | rpc_clnt_calls(3N) |
| rpc_call | rpc_clnt_calls(3N) |
| rpc_reg | rpc_svc_calls(3N) |
| svc_create | rpc_svc_create(3N) |
| svc_destroy | rpc_svc_create(3N) |
| svc_dg_create | rpc_svc_create(3N) |
| svc_dg_enablecache | rpc_svc_calls(3N) |
| svc_fd_create | rpc_svc_create(3N) |
| svc_fds | rpc_soc(3N) |
| svc_freeargs | rpc_svc_reg(3N) |
| svc_getargs | rpc_svc_reg(3N) |

| | |
|------------------|--------------------|
| svc_getcaller | rpc_soc(3N) |
| svc_getreq | rpc_soc(3N) |
| svc_getreqset | rpc_svc_calls(3N) |
| svc_getrpccaller | rpc_svc_calls(3N) |
| svc_kerb_reg | kerberos_rpc(3N) |
| svc_raw_create | rpc_svc_create(3N) |
| svc_reg | rpc_svc_calls(3N) |
| svc_register | rpc_soc(3N) |
| svc_run | rpc_svc_reg(3N) |
| svc_sendreply | rpc_svc_reg(3N) |
| svc_tli_create | rpc_svc_create(3N) |
| svc_tp_create | rpc_svc_create(3N) |
| svc_unreg | rpc_svc_calls(3N) |
| svc_unregister | rpc_soc(3N) |
| svc_vc_create | rpc_svc_create(3N) |
| svcerr_auth | rpc_svc_err(3N) |
| svcerr_decode | rpc_svc_err(3N) |
| svcerr_noproc | rpc_svc_err(3N) |
| svcerr_noprog | rpc_svc_err(3N) |
| svcerr_progvers | rpc_svc_err(3N) |
| svcerr_systemerr | rpc_svc_err(3N) |
| svcerr_weakauth | rpc_svc_err(3N) |
| svcfid_create | rpc_soc(3N) |

| | |
|--------------------|-------------------|
| svccraw_create | rpc_soc(3N) |
| svctcp_create | rpc_soc(3N) |
| svcudp_bufcreate | rpc_soc(3N) |
| svcudp_create | rpc_soc(3N) |
| user2netname | secure_rpc(3N) |
| xdr_accepted_reply | rpc_xdr(3N) |
| xdr_authsys_parms | rpc_xdr(3N) |
| xdr_authunix_parms | rpc_soc(3N) |
| xdr_callhdr | rpc_xdr(3N) |
| xdr_callmsg | rpc_xdr(3N) |
| xdr_opaque_auth | rpc_xdr(3N) |
| xdr_rejected_reply | rpc_xdr(3N) |
| xdr_replymsg | rpc_xdr(3N) |
| xprt_register | rpc_svc_calls(3N) |
| xprt_unregister | rpc_svc_calls(3N) |

FILES

/etc/netconfig

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |

SEE ALSO

getnetconfig(3N), **getnetpath(3N)**, **kerberos_rpc(3N)**,
rpc_clnt_auth(3N), **rpc_clnt_calls(3N)**, **rpc_clnt_create(3N)**,
rpc_svc_calls(3N), **rpc_svc_create(3N)**, **rpc_svc_err(3N)**,
rpc_svc_reg(3N), **rpc_xdr(3N)**, **rpcbind(3N)**, **secure_rpc(3N)**,
xdr(3N), **netconfig(4)**, **rpc(4)**, **attributes(5)**, **environ(5)**

| | |
|--------------------|---|
| NAME | rpcbind, rpcb_getmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – library routines for RPC bind service |
| SYNOPSIS | <pre>#include <rpc/rpc.h> struct rpcblist * rpcb_getmaps(const struct netconfig * <i>nnetconf</i>, const char * <i>host</i>); bool_t rpcb_getaddr(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>, struct netbuf * <i>svcaddr</i>, const char * <i>host</i>); bool_t rpcb_gettime(const char * <i>host</i>, time_t * <i>timep</i>); enum clnt_stat rpcb_rmtcall(const struct netconfig * <i>netconf</i>, const char * <i>host</i>, const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const rpcproc_t <i>procnum</i>, const xdrproc_t <i>inproc</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>, const struct timeval <i>tout</i>, struct netbuf * <i>svcaddr</i>); bool_t rpcb_set(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>, const struct netbuf * <i>svcaddr</i>); bool_t rpcb_unset(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>);</pre> |
| DESCRIPTION | <p>These routines allow client C programs to make procedure calls to the RPC binder service. rpcbind maintains a list of mappings between programs and their universal addresses. See rpcbind(1M) .</p> |
| Routines | <p>rpcb_getmaps()</p> <p>An interface to the <code>rpcbind</code> service, which returns a list of the current RPC program-to-address mappings on <i>host</i> . It uses the transport specified through <i>netconf</i> to contact the remote <code>rpcbind</code> service on <i>host</i> . This routine will return NULL, if the remote <code>rpcbind</code> could not be contacted.</p> <p>rpcb_getaddr()</p> <p>An interface to the <code>rpcbind</code> service, which finds the address of the service on <i>host</i> that is registered with program number <i>prognum</i> , version <i>versnum</i> , and speaks the transport protocol associated with <i>netconf</i> . The address found is returned in <i>svcaddr</i> . <i>svcaddr</i> should be preallocated. This routine returns TRUE if it succeeds. A return value of FALSE means that the mapping does not exist or that the RPC system failed to contact the remote <code>rpcbind</code> service. In the latter case, the global variable <code>rpc_createerr</code> contains the RPC status. See rpc_clnt_create(3N) .</p> <p>rpcb_gettime()</p> |

This routine returns the time on *host* in *timep*. If *host* is `NULL`, **rpcb_gettime()** returns the time on its own machine. This routine returns `TRUE` if it succeeds, `FALSE` if it fails. **rpcb_gettime()** can be used to synchronize the time between the client and the remote server. This routine is particularly useful for secure RPC.

rpcb_rmtcall()

An interface to the `rpcbind` service, which instructs `rpcbind` on *host* to make an RPC call on your behalf to a procedure on that host. The `netconfig` structure should correspond to a connectionless transport. The parameter **svcaddr* will be modified to the server's address if the procedure succeeds. See **rpc_call()** and **clnt_call()** in `rpc_clnt_calls(3N)` for the definitions of other parameters.

This procedure should normally be used for a "ping" and nothing else. This routine allows programs to do lookup and call, all in one step.

Note: Even if the server is not running `rpcbind` does not return any error messages to the caller. In such a case, the caller times out.

Note: **rpcb_rmtcall()** is only available for connectionless transports.

rpcb_set()

An interface to the `rpcbind` service, which establishes a mapping between the triple [*prognum*, *versnum*, *netconf* \Rightarrow *nc_netid*] and *svcaddr* on the machine's `rpcbind` service. The value of *nc_netid* must correspond to a network identifier that is defined by the `netconfig` database. This routine returns `TRUE` if it succeeds, `FALSE` otherwise. See also **svc_reg()** in `rpc_svc_calls(3N)`. If there already exists such an entry with `rpcbind`, **rpcb_set()** will fail.

rpcb_unset()

An interface to the `rpcbind` service, which destroys the mapping between the triple [*prognum*, *versnum*, *netconf* \Rightarrow *nc_netid*] and the address on the machine's `rpcbind` service. If *netconf* is `NULL`, **rpcb_unset()** destroys all mapping between the triple [*prognum*, *versnum*, *all-transport*] and the addresses on the machine's `rpcbind` service. This routine returns `TRUE` if it succeeds, `FALSE` otherwise. Only the owner of the service or the super-user can destroy the mapping. See also **svc_unreg()** in `rpc_svc_calls(3N)`.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`rpcbind(1M)` , `rpcinfo(1M)` , `rpc_clnt_calls(3N)` ,
`rpc_clnt_create(3N)` , `rpc_svc_calls(3N)` , `attributes(5)`

NAME | `rpc_clnt_auth`, `auth_destroy`, `authnone_create`, `authsys_create`,
`authsys_create_default` – library routines for client side remote procedure call authentication

DESCRIPTION | These routines are part of the RPC library that allows C language programs to make procedure calls on other machines across the network, with desired authentication.

These routines are normally called after creating the `CLIENT` handle. The `cl_auth` field of the `CLIENT` structure should be initialized by the `AUTH` structure returned by some of the following routines. The client's authentication information is passed to the server when the RPC call is made.

Only the `NULL` and the `SYS` style of authentication is discussed here. For the `DES` style authentication, please refer to `secure_rpc(3N)`. For the Kerberos style authentication, please refer to `kerberos_rpc(3N)`.

The `NULL` and `SYS` style of authentication are safe in multithreaded applications. For the MT-level of the `DES` and Kerberos styles, see their respective pages.

Routines | The following routines require that the header `<rpc/rpc.h>` be included (see `rpc(3N)` for the definition of the `AUTH` data structure).

```
#include <rpc/rpc.h>
```

```
void auth_destroy(AUTH * auth);
```

A function macro that destroys the authentication information associated with `auth`. Destruction usually involves deallocation of private data structures. The use of `auth` is undefined after calling `auth_destroy()`.

```
AUTH *authnone_create(void);
```

Create and return an RPC authentication handle that passes nonusable authentication information with each remote procedure call. This is the default authentication used by RPC.

```
AUTH *authsys_create(const char * host , const uid_t uid , const gid_t gid ,  
const int len , const gid_t * aup_gids);
```

Create and return an RPC authentication handle that contains `AUTH_SYS` authentication information. The parameter `host` is the name of the machine on which the information was created; `uid` is the user's user ID; `gid` is the user's current group ID; `len` and `aup_gids` refer to a counted array of groups to which the user belongs.

AUTH *aut hsys_create_default(void);

Call **authsys_create()** with the appropriate parameters.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

kerberos_rpc(3N) , **rpc(3N)** , **rpc_clnt_calls(3N)** ,
rpc_clnt_create(3N) , **secure_rpc(3N)** , **attributes(5)**

NAME | rpc_clnt_calls, clnt_call, clnt_freeres, clnt_geterr, clnt_perrno, clnt_perror, clnt_sperrno, clnt_sperror, rpc_broadcast, rpc_broadcast_exp, rpc_call – library routines for client side calls

DESCRIPTION | RPC library routines allow C language programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply.

The **clnt_call()**, **rpc_call()**, and **rpc_broadcast()** routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.

Some of the routines take a **CLIENT** handle as one of the parameters. A **CLIENT** handle can be created by an RPC creation routine such as **clnt_create()** (see **rpc_clnt_create(3N)**).

These routines are safe for use in multithreaded applications. **CLIENT** handles can be shared between threads, however in this implementation requests by different threads are serialized (that is, the first request will receive its results before the second request is sent).

Routines | See **rpc(3N)** for the definition of the **CLIENT** data structure.

```
#include <rpc/rpc.h>
```

```
enum clnt_stat clnt_call(CLIENT * clnt , const rpcproc_t procnum , const
xdrproc_t inproc , const caddr_t in , const xdrproc_t outproc , caddr_t out
, const struct timeval tout );
```

A function macro that calls the remote procedure *procnum* associated with the client handle, *clnt*, which is obtained with an RPC client creation routine such as **clnt_create()** (see **rpc_clnt_create(3N)**). The parameter *inproc* is the XDR function used to encode the procedure's parameters, and *outproc* is the XDR function used to decode the procedure's results; *in* is the address of the procedure's argument(s), and *out* is the address of where to place the result(s). *tout* is the time allowed for results to be returned, which is overridden by a time-out set explicitly through **clnt_control()**, see **rpc_clnt_create(3N)**.

If the remote call succeeds, the status returned is **RPC_SUCCESS**, otherwise an appropriate status is returned.

```
bool_t clnt_freeres(CLIENT * clnt , const xdrproc_t outproc , caddr_t out );
```

A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter *out* is the address of the results, and *outproc* is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.

void clnt_geterr(const CLIENT * *clnt* , struct rpc_err * *errp*);

A function macro that copies the error structure out of the client handle to the structure at address *errp* .

void clnt_perrno(const enum clnt_stat *stat*);

Print a message to standard error corresponding to the condition indicated by *stat* . A newline is appended. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()` .

void clnt_perror(const CLIENT * *clnt* , const char * *s*);

Print a message to the standard error indicating why an RPC call failed; *clnt* is the handle used to do the call. The message is prepended with string *s* and a colon. A newline is appended. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()` .

char *clnt_sperrno(const enum clnt_stat *stat*);

Take the same arguments as `clnt_perrno()` , but instead of sending a message to the standard error indicating why an RPC call failed, return a pointer to a string which contains the message.

`clnt_sperrno()` is normally used instead of `clnt_perrno()` when the program does not have a standard error (as a program running as a server quite likely does not), or if the programmer does not want the message to be output with `printf()` (see `printf(3S)`), or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcreateerror()` (see `rpc_clnt_create(3N)`), `clnt_sperrno()` does not return pointer to static data so the result will not get overwritten on each call.

char *clnt_sperror(const CLIENT * *clnt* , const char * *s*);

Like `clnt_perror()` , except that (like `clnt_sperrno()`) it returns a string instead of printing to standard error. However, `clnt_sperror()` does not append a newline at the end of the message.

Warning: returns pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
enum clnt_stat rpc_broadcast(const rpcprog_t prognum, const rpcvers_t
versnum, const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t
in, const xdrproc_t outproc, caddr_t out, const resultproc_t eachresult,
const char * nettype);
```

Like `rpc_call()`, except the call message is broadcast to all the connectionless transports specified by `nettype`. If `nettype` is `NULL`, it defaults to `"netpath"`. Each time it receives a response, this routine calls `eachresult()`, whose form is:

```
bool_t eachresult(caddr_t out, const struct netbuf *addr,
const struct netconfig *netconf);
```

where `out` is the same as `out` passed to `rpc_broadcast()`, except that the remote procedure's output is decoded there; `addr` points to the address of the machine that sent the results, and `netconf` is the netconfig structure of the transport on which the remote server responded. If `eachresult()` returns 0, `rpc_broadcast()` waits for more replies; otherwise it returns with appropriate status.

Warning: broadcast file descriptors are limited in size to the maximum transfer size of that transport. For Ethernet, this value is 1500 bytes.

`rpc_broadcast()` uses `AUTH_SYS` credentials by default (see `rpc_clnt_auth(3N)`).

```
enum clnt_stat rpc_broadcast_exp(const rpcprog_t prognum, const rpcvers_t
versnum, const rpcproc_t procnum, const xdrproc_t xargs, caddr_t argsp,
const xdrproc_t xresults, caddr_t resultsp, const resultproc_t eachresult,
const int inittime, const int waittime, const char * nettype);
```

Like `rpc_broadcast()`, except that the initial timeout, `inittime` and the maximum timeout, `waittime` are specified in milliseconds.

`inittime` is the initial time that `rpc_broadcast_exp()` waits before resending the request. After the first resend, the re-transmission interval increases exponentially until it exceeds `waittime`.

```
enum clnt_stat rpc_call(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_t inproc,
const char * in, const xdrproc_t outproc, char * out, const char * nettype);
```

Call the remote procedure associated with *prognum* , *versnum* , and *procnum* on the machine, *host* . The parameter *inproc* is used to encode the procedure's parameters, and *outproc* is used to decode the procedure's results; *in* is the address of the procedure's argument(s), and *out* is the address of where to place the result(s). *nettype* can be any of the values listed on [rpc\(3N\)](#) . This routine returns RPC_SUCCESS if it succeeds, or an appropriate status is returned. Use the [clnt_perrno\(\)](#) routine to translate failure status into error messages.

Warning: [rpc_call\(\)](#) uses the first available transport belonging to the class *nettype* , on which it can create a connection. You do not have control of timeouts or authentication using this routine.

ATTRIBUTES

See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

[printf\(3S\)](#) , [rpc\(3N\)](#) , [rpc_clnt_auth\(3N\)](#) , [rpc_clnt_create\(3N\)](#) , [attributes\(5\)](#)

NAME | rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_screateerror, clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr – library routines for dealing with creation and manipulation of CLIENT handles

DESCRIPTION | RPC library routines allow C language programs to make procedure calls on other machines across the network. First a CLIENT handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.

These routines are MT-Safe. In the case of multithreaded applications, the `_REENTRANT` flag must be defined on the command line at compilation time (`-D_REENTRANT`). When the `_REENTRANT` flag is defined, `rpc_createerr` becomes a macro which enables each thread to have its own `rpc_createerr`.

Routines | See `rpc(3N)` for the definition of the CLIENT data structure.

```
#include <rpc/rpc.h>
```

```
bool_t clnt_control(CLIENT * clnt , const uint_t req , char * info ) ;
```

A function macro to change or retrieve various information about a client object. *req* indicates the type of operation, and *info* is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of *req* and their argument types and what they do are:

```
CLSET_TIMEOUT struct timeval *          set total timeout
CLGET_TIMEOUT\011struct timeval *\011get total timeout
```

If the timeout is set using `clnt_control()`, the timeout argument passed by `clnt_call()` is ignored in all subsequent calls. If the timeout value is set to 0, `clnt_control()` immediately returns `RPC_TIMEDOUT`. Set the timeout parameter to 0 for batching calls.

```
CLGET_SERVER_ADDR struct netbuf * get server's address
CLGET_SVC_ADDR\011struct netbuf *\011get server's address
CLGET_FD\011int *\011get associated file descriptor
CLSET_FD_CLOSE\011void\011close the file descriptor when
\011\011destroying the client handle
\011\011(see
clnt_destroy()
)
CLSET_FD_NCLOSE\011void\011do not close the file
\011\011descriptor when destroying
```

the client handle

```

CLGET_VERS  rpcvers_t      get the RPC program's version
\011\011number associated with the
\011\011client handle
CLSET_VERS\011rpcvers_t\011set the RPC program's version
\011\011number associated with the
\011\011client handle. This assumes
\011\011that the RPC server for this
\011\011new version is still listening
\011\011at the address of the previous
\011\011version.
CLGET_XID\011uint32_t\011get the XID of the previous
\011\011remote procedure call
CLSET_XID\011uint32_t\011set the XID of the next
\011\011remote procedure call
CLGET_PROG\011rpcprog_t\011get program number
CLSET_PROG\011rpcprog_t\011set program number

```

CLIENT `clnt_create(const char * host, const rpcprog_t program, const rpcvers_t versnum, const char * nettype);`
 The following operations are valid for connectorless transports only.

```

CLSET_RETRY_TIMEOUT  struct timeval *    set the retry timeout
CLGET_RETRY_TIMEOUT  struct timeval *    get the retry timeout

```

The retry timeout is the time that RPC waits for the server to reply before retransmitting the request.

clnt_control() returns TRUE on success and FALSE on failure.

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class of transport protocol to use. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database.

clnt_create() tries all the transports of the *nettype* class available from the NETPATH environment variable and the netconfig database, and chooses the first successful one. A default timeout is set and can be modified using **clnt_control()**. This routine returns NULL if it fails. The **clnt_pcreateerror()** routine can be used to print the reason for failure.

Note: **clnt_create()** returns a valid client handle even if the particular version number supplied to **clnt_create()** is not registered with the rpcbind service. This mismatch will be discovered by a **clnt_call** later (see **clnt_call(3N)**).
 SunOS 5. (3N) Last modified 20 Feb 1998

```
CLIENT *clnt_create_timed(const char * host , const rpcprog_t prognum ,
const rpcvers_t versnum , const char * nettype , const struct timeval * timeout
);
```

Generic client creation routine which is similar to `clnt_create()` but which also has the additional parameter `timeout` that specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_timed()` call behaves exactly like the `clnt_create()` call.

```
CLIENT *clnt_create_vers(const char * host , const rpcprog_t prognum ,
rpcvers_t * vers_outp , const rpcvers_t vers_low , const rpcvers_t vers_high ,
char * nettype ) ;
```

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. `host` identifies the name of the remote host where the server is located. `nettype` indicates the class transport protocols to be used. If the routine is successful it returns a client handle created for the highest version between `vers_low` and `vers_high` that is supported by the server. `vers_outp` is set to this value. That is, after a successful return `vers_low <= *vers_outp <= vers_high`. If no version between `vers_low` and `vers_high` is supported by the server then the routine fails and returns NULL. A default timeout is set and can be modified using `clnt_control()`. This routine returns NULL if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the `rpcbind` service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`). However, `clnt_create_vers()` does this for you and returns a valid handle only if a version within the range supplied is supported by the server.

```
CLIENT *clnt_create_vers_timed(const char * host , const rpcprog_t prognum
, rpcvers_t * vers_outp , const rpcvers_t vers_low , const rpcvers_t vers_high
, char * nettype const struct timeval * timeout );
```

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter `timeout`, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

```
void clnt_destroy(CLIENT * clnt ) ;
```

A function macro that destroys the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling **clnt_destroy()**. If the RPC library opened the associated file descriptor, or `CLSET_FD_CLOSE` was set using **clnt_control()**, the file descriptor will be closed.

The caller should call `auth_destroy(clnt =>cl_auth)` (before calling **clnt_destroy()**) to destroy the associated AUTH structure (see **rpc_clnt_auth(3N)**).

```
CLIENT *clnt_dg_create(const int fildes , const struct netbuf * svcaddr ,
const rpcprog_t prognum , const rpcvers_t versnum , const uint_t sendsz ,
const uint_t recvsz ) ;
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connectionless transport. The remote program is located at address *svcaddr*. The parameter *fildes* is an open and bound file descriptor. This routine will resend the call message in intervals of 15 seconds until a response is received or until the call times out. The total time for the call to time out is specified by **clnt_call()** (see **clnt_call()** in **rpc_clnt_calls(3N)**). The retry time out and the total time out periods can be changed using **clnt_control()**. The user may set the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails.

```
void clnt_pcreateerror(const char * s ) ;
```

Print a message to standard error indicating why a client RPC handle could not be created. The message is prepended with the string *s* and a colon, and appended with a newline.

```
CLIENT *clnt_raw_create(const rpcprog_t prognum , const rpcvers_t
versnum );
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding RPC server should live in the same address space; (see **svc_raw_create()** in **rpc_svc_create(3N)**). This allows simulation of RPC and measurement of RPC overheads, such as round trip times, without any kernel or networking interference. This routine returns `NULL` if it fails. **clnt_raw_create()** should be called after **svc_raw_create()**.

```
char *clnt_screateerror(const char * s );
```

Like `clnt_pcreateerror()` , except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.

Warning: returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
CLIENT *clnt_tli_create(const int fildev , const struct netconfig * netconf ,
const struct netbuf * svcaddr , const rpcprog_t prognum , const rpcvers_t
versnum , const uint_t sendsz , const uint_t rcvsvsz );
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum* . The remote program is located at address *svcaddr* . If *svcaddr* is `NULL` and it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is `NULL` , `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be open, bound and connected. If it is `RPC_ANYFD` , it opens a file descriptor on the transport specified by *netconf* . If *fildev* is `RPC_ANYFD` and *netconf* is `NULL` , a `RPC_UNKNOWNPROTO` error is set. If *fildev* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *rcvsvsz* ; values of 0 choose suitable defaults.

Depending upon the type of the transport (connection-oriented or connectionless), `clnt_tli_create()` calls appropriate client creation routines. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure. The remote `rpcbind` service (see `rpcbind(1M)`) is not consulted for the address of the remote service.

```
CLIENT *clnt_tp_create(const char * host , const rpcprog_t prognum , const
rpcvers_t versnum , const struct netconfig * netconf );
```

Like `clnt_create()` except `clnt_tp_create()` tries only one transport specified through *netconf* .

`clnt_tp_create()` creates a client handle for the program *prognum* , the version *versnum* , and for the transport specified by *netconf* . Default options are set, which can be changed using `clnt_control()` calls. The remote `rpcbind` service on the host *host* is consulted for the address of the remote service. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

```
CLIENT *clnt_tp_create_timed(const char * host , const rpcprog_t prognum ,
const rpcvers_t versnum , const struct netconfig * netconf , const struct
timeval * timeout );
```

Like `clnt_tp_create()` except `clnt_tp_create_timed()` has the extra parameter *timeout* which specifies the maximum time allowed for the creation attempt

to succeed. In all other respects, the `clnt_tp_create_timed()` call behaves exactly like the `clnt_tp_create()` call.

CLIENT `*clnt_vc_create(const int fildev, const struct netbuf * svcaddr, const rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t recvsz);`

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connection-oriented transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails.

The address *svcaddr* should not be `NULL` and should point to the actual address of the remote program. `clnt_vc_create()` does not consult the remote `rpcbind` service for this information.

struct `rpc_createerr` `rpc_createerr;`

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

In multithreaded applications, `rpc_createerr` becomes a macro which enables each thread to have its own `rpc_createerr`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`rpcbind(1M)`, `rpc(3N)`, `rpc_clnt_auth(3N)`, `rpc_clnt_calls(3N)`, `rpc_svc_create(3N)`, `svc_raw_create(3N)`, `attributes(5)`

NAME | `rpc_control` – library routine for manipulating global RPC attributes for client and server applications

SYNOPSIS | `bool_t rpc_control(int op, void *info);`

DESCRIPTION | This RPC library routine allows applications to set and modify global RPC attributes that apply to clients as well as servers. At present, it supports only server side operations. This function allows applications to set and modify global attributes that apply to client as well as server functions. *op* indicates the type of operation, and *info* is a pointer to the operation specific information. The supported values of *op* and their argument types, and what they do are:

```
RPC_SVC_MTMODE_SET int * set multithread mode
RPC_SVC_MTMODE_GET int * get multithread mode
RPC_SVC_THRMAX_SET int * set maximum number of threads
RPC_SVC_THRMAX_GET int * get maximum number of threads
RPC_SVC_THRTOTAL_GET int * get number of active threads
RPC_SVC_THRCREATES_GET int * get number of threads created
RPC_SVC_THRERRORS_GET int * get number of thread create errors
RPC_SVC_USE_POLLFD int * set number of file descriptors to unlimited
```

There are three multithread (MT) modes. These are:

```
RPC_SVC_MT_NONE Single threaded mode (default)
RPC_SVC_MT_AUTO Automatic MT mode
RPC_SVC_MT_USER User MT mode
```

Unless the application sets the Automatic or User MT modes, it will stay in the default (single threaded) mode. See the Network Interfaces Programming Guide for the meanings of these modes and programming examples. Once a mode is set, it cannot be changed.

By default, the maximum number of threads that the server will create at any time is 16. This allows the service developer to put a bound on thread resources consumed by a server. If a server needs to process more than 16 client requests concurrently, the maximum number of threads must be set to the desired number. This parameter may be set at any time by the server.

Set and get operations will succeed even in modes where the operations don't apply. For example, you can set the maximum number of threads in any mode, even though it makes sense only for the Automatic MT mode. All of the get operations except `RPC_SVC_MTMODE_GET` apply only to the Automatic MT mode, so values returned in other modes may be undefined.

By default, RPC servers are limited to a maximum of 1024 file descriptors or connections due to limitations in the historical interfaces `svc_fdset(3N)` and `svc_getreqset(3N)`. Applications written to use the preferred interfaces of `svc_pollfd(3N)` and `svc_getreq_poll(3N)` can use an unlimited number

of file descriptors. Setting `info` to point to a non-zero integer and `op` to `RPC_SVC_USE_POLLFD` removes the limitation.

RETURN VALUES

This routine returns `TRUE` if the operation was successful, and `FALSE` otherwise.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`rpcbind(1M)`, `rpc(3N)`, `rpc_svc_calls(3N)`, `attributes(5)`

| | |
|----------------------|--|
| NAME | rpc_gss_getcred – get credentials of client |
| SYNOPSIS | <pre>#include <rpc/rpcsec_gss.h> bool_t rpc_gss_getcred(struct svc_req *req, rpc_gss_raw_cred_t **rcred, rpc_gss_ucred **ucred, void **cookie);</pre> |
| DESCRIPTION | <p>rpc_gss_getcred() is used by a server to fetch the credentials of a client. These credentials may either be network credentials (in the form of a <code>rpc_gss_rawcred_t</code> structure) or UNIX credentials.</p> <p>For more information on RPCSEC_GSS data types, see the <code>rpcsec(3N)</code> man page.</p> |
| PARAMETERS | <p>Essentially, rpc_gss_getcred() passes a pointer to a request (<code>svc_req</code>) as well as pointers to two credential structures and a user-defined cookie; if rpc_gss_getcred() is successful, at least one credential structure is "filled out" with values, as is, optionally, the cookie.</p> <p>req Pointer to the received service request. <code>svc_req</code> is an RPC structure containing information on the context of an RPC invocation, such as program, version, and transport information.</p> <p>rcred A pointer to an <code>rpc_gss_rawcred_t</code> structure pointer. This structure contains the version number of the RPCSEC_GSS protocol being used; the security mechanism and QOPs for this session (as strings); principal names for the client (as a <code>rpc_gss_principal_t</code> structure) and server (as a string); and the security service (integrity, privacy, etc., as an enum). If an application is not interested in these values, it may pass <code>NULL</code> for this parameter.</p> <p>ucred The caller's UNIX credentials, in the form of a pointer to a pointer to a <code>rpc_gss_ucred_t</code> structure, which includes the client's uid and gids. If an application is not interested in these values, it may pass <code>NULL</code> for this parameter.</p> <p>cookie A four-byte quantity that an application may use in any manner it wants to; RPC does not interpret it. (For example, a cookie may be a pointer or index to a structure that represents a context initiator.) See also <code>rpc_gss_set_callback(3N)</code>.</p> |
| RETURN VALUES | <p>rpc_gss_getcred() returns <code>TRUE</code> if it is successful; otherwise, use rpc_gss_get_error() to get the error associated with the failure.</p> |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------|
| MT-Level | MT-Safe |
| Packages | SUNWrsg, SUNWrsgx |

SEE ALSO

rpc(3N), **rpc_gss_set_callback(3N)**, **rpc_gss_set_svc_name(3N)**,
rpcsec_gss(3N), **attributes(5)**

ONC+ Developer's Guide

Network Working Group RFC 2078

| NAME | rpc_gss_get_error – get error codes on failure | | | | | | |
|----------------------|--|----------------|-----------------|----------|---------|----------|-------------------|
| SYNOPSIS | <pre>#include <rpc/rpcsec_gss.h> bool_t rpc_gss_get_error(rpc_gss_error_t*error);</pre> | | | | | | |
| DESCRIPTION | <p>rpc_gss_get_error() fetches an error code when an RPCSEC_GSS routine fails.</p> <p>rpc_gss_get_error() uses a <code>rpc_gss_error_t</code> structure of the following form:</p> <pre>typedef struct { int rpc_gss_error; <i>RPCSEC_GSS error</i> int system_error; <i>system error</i> } rpc_gss_error_t;</pre> <p>Currently the only error codes defined for this function are</p> <pre>#define RPC_GSS_ER_SUCCESS 0 /* no error */ #define RPC_GSS_ER_SYSTEMERROR 1 /* system error */</pre> | | | | | | |
| PARAMETERS | <p>Information on RPCSEC_GSS data types for parameters may be found on the <code>rpcsec_gss(3N)</code> man page.</p> <p><code>error</code> A <code>rpc_gss_error_t</code> structure. If the <code>rpc_gss_error</code> field is equal to <code>RPC_GSS_ER_SYSTEMERROR</code>, the <code>system_error</code> field will be set to the value of <code>errno</code>.</p> | | | | | | |
| RETURN VALUES | Unless there is a failure indication from an invoked RPCSEC_GSS function, rpc_gss_get_error() does not set <code>error</code> to a meaningful value. | | | | | | |
| ATTRIBUTES | See <code>attributes(5)</code> for descriptions of the following attributes: | | | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> <tr> <td>Packages</td> <td>SUNWrsg, SUNWrsgx</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe | Packages | SUNWrsg, SUNWrsgx |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| MT-Level | MT-Safe | | | | | | |
| Packages | SUNWrsg, SUNWrsgx | | | | | | |
| SEE ALSO | <p><code>pererror(3C)</code>, <code>rpc(3N)</code>, <code>rpcsec_gss(3N)</code>, <code>attributes(5)</code></p> <p><i>ONC+ Developer's Guide</i></p> | | | | | | |

rpc_gss_get_error(3N)

Network Functions

NOTES

Network Working Group RFC 2078

Only system errors are currently returned.

| | |
|--------------------|--|
| NAME | rpc_gss_get_mechanisms, rpc_gss_get_mech_info, rpc_gss_get_versions, rpc_gss_is_installed – get information on mechanisms and RPC version |
| SYNOPSIS | <pre>#include <rpc/rpcsec_gss.h> char ** rpc_gss_get_mechanisms(); char ** rpc_gss_get_mech_info(char * mech, rpc_gss_service_t * service); bool_t rpc_gss_get_versions(u_int * vers_hi, u_int * vers_lo); bool_t rpc_gss_is_installed(char * mech);</pre> |
| DESCRIPTION | <p>These "convenience functions" return information on available security mechanisms and versions of RPCSEC_GSS.</p> <p>rpc_gss_get_mechanisms() Returns a list of supported security mechanisms as a null-terminated list of character strings.</p> <p>rpc_gss_get_mech_info() Takes two arguments: an ASCII string representing a mechanism type (e.g., "kerberosv5") and a pointer to a <code>rpc_gss_service_t</code> enum. Returns a null-terminated list of character strings of supported Quality of Protections (QOPs) for this mechanism.</p> <p>rpc_gss_get_versions() Returns the highest and lowest versions of RPCSEC_GSS supported.</p> <p>rpc_gss_is_installed() Takes an ASCII string representing a mechanism, and returns <code>TRUE</code> if the mechanism is installed.</p> |
| PARAMETERS | <p>Information on RPCSEC_GSS data types for parameters may be found on the <code>rpcsec_gss(3N)</code> man page.</p> <p>mech An ASCII string representing the security mechanism in use. Valid strings may also be found in the <code>/etc/gss/mech</code> file.</p> <p>service A pointer to a <code>rpc_gss_service_t</code> enum, representing the current security service (privacy, integrity, or none).</p> <p>vers_hi</p> <p>vers_lo The highest and lowest versions of RPCSEC_GSS supported.</p> |
| FILES | <pre>/etc/gss/mech File containing valid security mechanisms /etc/gss/qop File containing valid QOP values</pre> |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------|
| MT-Level | MT-Safe |
| Packages | SUNWrsg, SUNWrsgx |

SEE ALSO

rcp(3N) , **rcpsec_gss(3N)** , **mehc(4)** , **qop(4)** , **attributes(5)**

ONC+ Developer's Guide

Network Working Group RFC 2078

NOTES

This function will change in a future release.

BUGS

The *service* argument for **rpc_gss_get_mech_info()** is currently irrelevant.

| | |
|--------------------|---|
| NAME | rpc_gss_get_principal_name – Get principal names at server |
| SYNOPSIS | <pre>#include <rpc/rpcsec_gss.h> bool_t rpc_gss_get_principal_name(rpc_gss_principal_ *principal, char *mech, char *name, char *node, char *domain);</pre> |
| DESCRIPTION | <p>Servers need to be able to operate on a client's principal name. Such a name is stored by the server as a <code>rpc_gss_principal_t</code> structure, an opaque byte string which can be used either directly in access control lists or as database indices which can be used to look up a UNIX credential. A server may, for example, need to compare a principal name it has received with the principal name of a known entity, and to do that, it must be able to generate <code>rpc_gss_principal_t</code> structures from known entities.</p> <p>rpc_gss_get_principal_name() takes as input a security mechanism, a pointer to a <code>rpc_gss_principal_t</code> structure, and several parameters which uniquely identify an entity on a network: a user or service name, a node name, and a domain name. From these parameters it constructs a unique, mechanism-dependent principal name of the <code>rpc_gss_principal_t</code> structure type.</p> |
| PARAMETERS | <p>How many of the identifying parameters (<i>name</i>, <i>node</i>, and <i>domain</i>) are necessary to specify depends on the mechanism being used. For example, Kerberos V5 requires only a user name but can accept a node and domain name. An application can choose to set unneeded parameters to <code>NULL</code>.</p> <p>Information on <code>RPCSEC_GSS</code> data types for parameters may be found on the <code>rpcsec_gss(3N)</code> man page.</p> <p><i>principal</i> An opaque, mechanism-dependent structure representing the client's principal name.</p> <p><i>mech</i> An ASCII string representing the security mechanism in use. Valid strings may be found in the <code>/etc/gss/mech</code> file, or by using rpc_gss_get_mechanisms().</p> <p><i>name</i> A UNIX login name (for example, 'gswashington') or service name, such as 'nfs'.</p> <p><i>node</i> A node in a domain; typically, this would be a machine name (for example, 'valleyforge').</p> <p><i>domain</i> A security domain; for example, a DNS, NIS, or NIS+ domain name ('eng.company.com').</p> |

RETURN VALUES

rpc_gss_get_principal_name() returns TRUE if it is successful; otherwise, use **rpc_gss_get_error()** to get the error associated with the failure.

FILES

/etc/gss/mech File containing valid security mechanisms

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------|
| MT-Level | MT-Safe |
| Packages | SUNWrsg, SUNWrsgx |

SEE ALSO

free(3C), **rpc(3N)**, **rpc_gss_get_mechanisms(3N)**,
rpc_gss_set_svc_name(3N), **rpcsec_gss(3N)**, **mech(4)**, **attributes(5)**

ONC+ Developer's Guide

Network Working Group RFC 2078

NOTES

Principal names may be freed up by a call to **free(3C)**. A principal name need only be freed in those instances where it was constructed by the application. (Values returned by other routines point to structures already existing in a context, and need not be freed.)

| NAME | rpc_gss_max_data_length, rpc_gss_svc_max_data_length – get maximum data length for transmission | | | | | | |
|------------------------|---|----------------|--|------------------------|---|------------|--|
| SYNOPSIS | <pre>#include <rpc/rpcsec_gss.h> int rpc_gss_max_data_length(AUTH *handle, int max_tp_unit_len); int rpc_gss_svc_max_data_length(struct svc_req *req, int max_tp_unit_len);</pre> | | | | | | |
| DESCRIPTION | <p>Performing a security transformation on a piece of data generally produces data with a different (usually greater) length. For some transports, such as UDP, there is a maximum length of data which can be sent out in one data unit. Applications need to know the maximum size a piece of data can be before it's transformed, so that the resulting data will still "fit" on the transport. These two functions return that maximum size.</p> <p>rpc_gss_max_data_length() is the client-side version; rpc_gss_svc_max_data_length() is the server-side version.</p> | | | | | | |
| PARAMETERS | <table border="0"> <tr> <td style="padding-right: 20px;">handle</td> <td>An RPC context handle of type <code>AUTH</code>, returned when a context is created (for example, by rpc_gss_seccreate()). Security service and QOP are bound to this handle, eliminating any need to specify them.</td> </tr> <tr> <td style="padding-right: 20px;">max_tp_unit_len</td> <td>The maximum size of a piece of data allowed by the transport.</td> </tr> <tr> <td style="padding-right: 20px;">req</td> <td>A pointer to an RPC <code>svc_req</code> structure, containing information on the context (for example, program number and credentials).</td> </tr> </table> | handle | An RPC context handle of type <code>AUTH</code> , returned when a context is created (for example, by rpc_gss_seccreate()). Security service and QOP are bound to this handle, eliminating any need to specify them. | max_tp_unit_len | The maximum size of a piece of data allowed by the transport. | req | A pointer to an RPC <code>svc_req</code> structure, containing information on the context (for example, program number and credentials). |
| handle | An RPC context handle of type <code>AUTH</code> , returned when a context is created (for example, by rpc_gss_seccreate()). Security service and QOP are bound to this handle, eliminating any need to specify them. | | | | | | |
| max_tp_unit_len | The maximum size of a piece of data allowed by the transport. | | | | | | |
| req | A pointer to an RPC <code>svc_req</code> structure, containing information on the context (for example, program number and credentials). | | | | | | |
| RETURN VALUES | Both functions return the maximum size of untransformed data allowed, as an <code>int</code> . | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> <tr> <td>Packages</td> <td>SUNWrsg, SUNWrsgx</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe | Packages | SUNWrsg, SUNWrsgx |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| MT-Level | MT-Safe | | | | | | |
| Packages | SUNWrsg, SUNWrsgx | | | | | | |
| SEE ALSO | <p><code>rpc(3N)</code>, <code>rpcsec_gss(3N)</code>, <code>attributes(5)</code></p> <p><i>ONC+ Developer's Guide</i></p> | | | | | | |

rpc_gss_max_data_length(3N)

Network Functions

Network Working Group RFC 2078

| | |
|----------------------|--|
| NAME | rpc_gss_mech_to_oid, rpc_gss_qop_to_num – map mechanism, QOP strings to non-string values |
| SYNOPSIS | <pre>#include <rpc/rpcsec_gss.h> bool_t rpc_gss_mech_to_oid(charc *mech, rpc_gss_OIDc *oid); bool_t rpc_gss_qop_to_num(char *qop, char *mech, u_int *num);</pre> |
| DESCRIPTION | <p>Because in-kernel RPC routines use non-string values for mechanism and Quality of Protection (QOP), these routines exist to map strings for these attributes to their non-string counterparts. (The non-string values for QOP and mechanism are also found in the <code>/etc/gss/qop</code> and <code>/etc/gss/mech</code> files, respectively.) rpc_gss_mech_to_oid() takes a string representing a mechanism, as well as a pointer to a <code>rpc_gss_OID</code> object identifier structure. It then gives this structure values corresponding to the indicated mechanism, so that the application can now use the OID directly with RPC routines. rpc_gss_qop_to_num() does much the same thing, taking strings for QOP and mechanism and returning a number.</p> |
| PARAMETERS | <p>Information on <code>RPCSEC_GSS</code> data types for parameters may be found on the <code>rpcsec_gss(3N)</code> man page.</p> <p>mech An ASCII string representing the security mechanism in use. Valid strings may be found in the <code>/etc/gss/mech</code> file.</p> <p>oid An object identifier of type <code>rpc_gss_OID</code>, whose elements are usable by kernel-level RPC routines.</p> <p>qop This is an ASCII string which sets the quality of protection (QOP) for the session. Appropriate values for this string may be found in the file <code>/etc/gss/qop</code>.</p> <p>num The non-string value for the QOP.</p> |
| RETURN VALUES | Both functions return TRUE if they are successful, FALSE otherwise. |
| FILES | <pre>/etc/gss/mech File containing valid security mechanisms /etc/gss/qop File containing valid QOP values</pre> |
| ATTRIBUTES | See <code>attributes(5)</code> for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------|
| MT-Level | MT-Safe |
| Packages | SUNWrsg, SUNWrsgx |

SEE ALSO

`rpc(3N)`, `rpc_gss_get_error(3N)`, `rpc_gss_get_mechanism(3N)`,
`rpcsec_gss(3N)`, `mech(4)`, `qop(4)`, `attributes(5)`

ONC+ Developer's Guide

Network Working Group RFC 2078

| | |
|--------------------|---|
| NAME | rpc_gss_seccreate – create a security context using the |
| SYNOPSIS | <pre>#include <rpc/rpcsec_gss.h> AUTH *rpc_gss_seccreate(CLIENT *clnt, char *principal, char *mechanism, rpc_gss_service_t service_type, char *qop, rpc_gss_options_req_t *options_req, rpc_gss_options_ret_t *options_ret);</pre> |
| DESCRIPTION | <p>rpc_gss_seccreate() is used by an application to create a security context using the <code>RPCSEC_GSS</code> protocol, making use of the underlying <code>GSS_API</code> network layer. rpc_gss_seccreate() allows an application to specify the type of security mechanism (for example, Kerberos v5), the type of service (for example, integrity checking), and the Quality of Protection (QOP) desired for transferring data.</p> |
| PARAMETERS | <p>Information on <code>RPCSEC_GSS</code> data types for parameters may be found on the <code>rpcsec_gss(3N)</code> man page.</p> <p>clnt This is the RPC client handle. <i>clnt</i> may be obtained, for example, from <code>clnt_create()</code>.</p> <p>principal This is the identity of the server principal, specified in the form <i>service@host</i>, where <i>service</i> is the name of the service the client wishes to access and <i>host</i> is the fully qualified name of the host where the service resides — for example, <code>nfs@mymachine.eng.company.com</code>.</p> <p>mechanism This is an ASCII string which indicates which security mechanism to use with this data. Appropriate mechanisms may be found in the file <code>/etc/gss/mech</code>; additionally, rpc_gss_get_mechanisms() returns a list of supported security mechanisms (as null-terminated strings).</p> <p>service_type This sets the initial type of service for the session — privacy, integrity, authentication, or none.</p> <p>qop This is an ASCII string which sets the quality of protection (QOP) for the session. Appropriate values for this string may be found in the file <code>/etc/gss/qop</code>. Additionally, supported QOPs are returned (as null-terminated strings) by rpc_gss_get_mech_info().</p> |

options_req This structure contains options which are passed directly to the underlying GSS-API layer. If the caller specifies NULL for this parameter, defaults are used. (See NOTES, below.)

options_ret These GSS-API options are returned to the caller. If the caller does not need to see these options, then it may specify NULL for this parameter. (See NOTES, below.)

RETURN VALUES

rpc_gss_seccreate() returns a security context handle (an RPC authentication handle) of type AUTH. If **rpc_gss_seccreate()** cannot return successfully, the application can get an error number by calling **rpc_gss_get_error()**.

FILES

/etc/gss/mech File containing valid security mechanisms
 /etc/gss/qop File containing valid QOP values .

ATTRIBUTES

See **attributes(5N)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------|
| MT-Level | MT-Safe |
| Packages | SUNWrsg, SUNWrsgx |

SEE ALSO

auth_destroy(3N), **rpc(3N)**, **rpc_gss_get_error(3N)**,
rpc_gss_get_mechanism(3N), **rpcsec_gss(3N)**, **mech(4)**, **qop(4)**,
attributes(5)

ONC+ Developer's Guide

Network Working Group RFC 2078

NOTES

Contexts may be destroyed normally, with **auth_destroy()**.

Currently, the GSS-API interface is not exposed. Therefore, use NULL for both the **options_req** and **options_ret** parameters.

| | |
|--------------------|--|
| NAME | rpc_gss_set_callback – specify callback for context |
| SYNOPSIS | <pre>#include <rpc/rpcsec_gss.h> bool_t rpc_gss_set_callback(struct rpc_gss_callback_t *cb);</pre> |
| DESCRIPTION | <p>A server may want to specify a callback routine so that it knows when a context gets first used. This user-defined callback may be specified through the rpc_gss_set_callback() routine. The callback routine is invoked the first time a context is used for data exchanges, after the context is established for the specified program and version.</p> <p>The user-defined callback routine should take the following form:</p> <pre>bool_t callback(struct svc_req *req, gss_cred_id_t deleg, gss_ctx_id_t gss_context, rpc_gss_lock_t *lock, void **cookie);</pre> |
| PARAMETERS | <p>rpc_gss_set_callback() takes one argument: a pointer to a <code>rpc_gss_callback_t</code> structure. This structure contains the RPC program and version number as well as a pointer to a user-defined callback() routine. (For a description of <code>rpc_gss_callback_t</code> and other <code>RPCSEC_GSS</code> data types, see the <code>rpcsec(3N)</code> man page.)</p> <p>The user-defined callback() routine itself takes the following arguments:</p> <p>req Pointer to the received service request. <code>svc_req</code> is an RPC structure containing information on the context of an RPC invocation, such as program, version, and transport information.</p> <p>deleg Delegated credentials, if any. (See <code>NOTES</code>, below.)</p> <p>gss_context GSS context (allows server to do GSS operations on the context to test for acceptance criteria). (See <code>NOTES</code>, below.)</p> <p>lock This parameter is used to enforce a particular QOP and service for a session. This parameter points to a <code>RPCSEC_GSS rpc_gss_lock_t</code> structure. When the callback is invoked, the <code>rpc_gss_lock_t.locked</code> field is set to <code>TRUE</code>, thus locking the context. A locked context will reject all requests having different values for QOP or service than those specified by the <code>raw_cred</code> field of the <code>rpc_gss_lock_t</code> structure.</p> |

cookie A four-byte quantity that an application may use in any manner it wants to — RPC does not interpret it. (For example, the cookie could be a pointer or index to a structure that represents a context initiator.) The cookie is returned, along with the caller's credentials, with each invocation of **rpc_gss_getcred()**.

RETURN VALUES

rpc_gss_set_callback() returns TRUE if the use of the context is accepted; false otherwise.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------|
| MT-Level | MT-Safe |
| Packages | SUNWrsg, SUNWrsgx |

SEE ALSO

rpc(3N), **rpc_gss_getcred(3N)**, **rpcsec_gss(3N)**, **attributes(5)**

ONC+ Developer's Guide

Network Working Group RFC 2078

NOTES

If a server does not specify a callback, all incoming contexts will be accepted.

Because the GSS-API is not currently exposed, the *deleg* and *gss_context* arguments are mentioned for informational purposes only, and the user-defined callback function may choose to do nothing with them.

| NAME | rpc_gss_set_defaults – change service, QOP for a session | | | | | | |
|----------------------|---|----------------|-----------------|----------|---------|----------|-------------------|
| SYNOPSIS | <pre>#include <rpc/rpcsec_gss.h> bool_t rpc_gss_set_defaults(AUTH *auth, rpc_gss_service_t service, char *qop);</pre> | | | | | | |
| DESCRIPTION | rpc_gss_set_defaults() allows an application to change the service (privacy, integrity, authentication, or none) and Quality of Protection (QOP) for a transfer session. New values apply to the rest of the session (unless changed again). | | | | | | |
| PARAMETERS | <p>Information on <code>RPCSEC_GSS</code> data types for parameters may be found on the <code>rpcsec_gss(3N)</code> man page.</p> <p>auth An RPC authentication handle returned by <code>rpc_gss_seccreate()</code>.</p> <p>service An enum of type <code>rpc_gss_service_t</code>, representing one of the following types of security service: authentication, privacy, integrity, or none.</p> <p>qop A string representing Quality of Protection. Valid strings may be found in the file <code>/etc/gss/qop</code> or by using <code>rpc_gss_get_mech_info()</code>.</p> | | | | | | |
| RETURN VALUES | <code>rpc_gss_set_svc_name()</code> returns TRUE if it is successful; otherwise, use <code>rpc_gss_get_error()</code> to get the error associated with the failure. | | | | | | |
| FILES | <code>/etc/gss/qop</code> File containing valid QOPs | | | | | | |
| ATTRIBUTES | See <code>attributes(5)</code> for descriptions of the following attributes: | | | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> <tr> <td>Packages</td> <td>SUNWrsg, SUNWrsgx</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe | Packages | SUNWrsg, SUNWrsgx |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| MT-Level | MT-Safe | | | | | | |
| Packages | SUNWrsg, SUNWrsgx | | | | | | |
| SEE ALSO | <p><code>rpc(3N)</code>, <code>rpc_gss_get_mech_info(3N)</code>, <code>rpcsec_gss(3N)</code>, <code>qop(4)</code>, <code>attributes(5)</code></p> <p><i>ONC+ Developer's Guide</i></p> <p><i>Network Working Group RFC 2078</i></p> | | | | | | |

| NAME | rpc_gss_set_svc_name – send a principal name to a server | | | | | | |
|----------------------|--|----------------|-----------------|----------|---------|----------|-------------------|
| SYNOPSIS | <pre>#include <rpc/rpcsec_gss.h> bool_t rpc_gss_set_svc_name(char *principal, char *mechanism, u_int req_time, u_int program, u_int version);</pre> | | | | | | |
| DESCRIPTION | rpc_gss_set_svc_name() sets the name of a principal the server is to represent. If a server is going to act as more than one principal, this procedure can be invoked for every such principal. | | | | | | |
| PARAMETERS | <p>Information on RPCSEC_GSS data types for parameters may be found on the <code>rpcsec_gss(3N)</code> man page.</p> <p>principal An ASCII string representing the server's principal name, given in the form of <i>service@host</i>.</p> <p>mech An ASCII string representing the security mechanism in use. Valid strings may be found in the <code>/etc/gss/mech</code> file, or by using <code>rpc_gss_get_mechanisms()</code>.</p> <p>req_time The time, in seconds, for which a credential should be valid.</p> <p>program The RPC program number for this service.</p> <p>version The RPC version number for this service.</p> | | | | | | |
| RETURN VALUES | rpc_gss_set_svc_name() returns TRUE if it is successful; otherwise, use <code>rpc_gss_get_error()</code> to get the error associated with the failure. | | | | | | |
| FILES | <code>/etc/gss/mech</code> File containing valid security mechanisms | | | | | | |
| ATTRIBUTES | See <code>attributes(5)</code> for descriptions of the following attributes: | | | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> <tr> <td>Packages</td> <td>SUNWrsg, SUNWrsgx</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe | Packages | SUNWrsg, SUNWrsgx |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| MT-Level | MT-Safe | | | | | | |
| Packages | SUNWrsg, SUNWrsgx | | | | | | |
| SEE ALSO | <p><code>rpc(3N)</code>, <code>rpc_gss_get_mechanisms(3N)</code>, <code>rpc_gss_get_principal_name(3N)</code>, <code>rpcsec_gss(3N)</code>, <code>mech(4)</code>, <code>attributes(5)</code></p> <p><i>ONC+ Developer's Guide</i></p> | | | | | | |

Network Working Group RFC 2078

| | |
|--------------------|---|
| NAME | rpc_rac, rac_drop, rac_poll, rac_recv, rac_send – remote asynchronous calls |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lrac -lnsl [<i>library</i> ...] #include <rpc/rpc.h> #include <rpc/rac.h> void rac_drop(CLIENT *cl, void *h); enum clnt_stat rac_poll(CLIENT *cl, void *h); enum clnt_stat rac_recv(CLIENT *cl, void *h); void *rac_send(CLIENT *cl, rpcproc_t proc, xdrproc_t xargs, void *argsp, xdrproc_t xresults, void *resultsp, struct timeval timeout); </pre> |
| DESCRIPTION | <p>The remote asynchronous calls (RAC) package is a special interface to the RPC library that allows messages to be sent using the RPC protocol without blocking during the time between when the message is sent and the reply is received. To RPC servers, RAC messages are indistinguishable from RPC messages.</p> <p>A client establishes an RPC session in the usual way (see rpc_clnt_create(3N)). A RAC message is sent using rac_send() . This routine returns immediately, allowing the client to conduct other processing. When the client wants to determine whether the returned value from the call has been received, rac_poll() is used. rac_recv() is used to collect the returned value; it can also be used to block while waiting for the returned value to arrive. rac_drop() is used to inform the RPC library that the client is no longer interested in the results of a particular RAC message.</p> <p>rac_drop() rac_drop() should be called when the user is no longer interested in the result of a rac_send() currently in progress. No message to the server is generated by this call, but any subsequent reply received for this handle will be silently dropped. It also frees any space occupied by the asynchronous call handle <i>h</i> .</p> |

After a call to **rac_drop()** the handle referred to by *h* is invalid. It may no longer be used in any asynchronous operation.

rac_poll()

rac_poll() returns the status of the call currently in progress on the <CLIENT, asynchronous handle> tuple referred to by *cl* and *h*.

rac_poll() return values are:

| | |
|--------------------|--|
| RPC_SUCCESS | A reply has been received and is available for reading by rac_rcv() . |
| RPC_INPROGRESS | No reply has been received. The call referred to by the given handle has not yet timed out. |
| RPC_TIMEDOUT | No reply has been received. The call referred to by the given handle has exceeded the maximum timeout value specified in rac_send() . |
| RPC_STALERACHANDLE | Either the handle referred to by <i>h</i> is invalid or no call is currently in progress for the given <CLIENT, asynchronous handle> tuple. |
| RPC_CANTRECV | Either the file descriptor associated with the given CLIENT handle is bad, or an error occurred while attempting to receive a packet. |
| RPC_SYSTEMERROR | Space could not be allocated to receive a packet. |

On unreliable transports, a call to **rac_poll()** will trigger a retransmission when necessary (that is, if a **rac_send()** is in progress, no reply has been received, the per-call timeout has expired, and the total timeout has not yet expired).

The return value for **rac_poll()** is independent of the RPC return value in the reply packet. Although a combination of **clnt_control()**'s CLGET_FD request and **poll(2)** may be used to extract the proper file descriptor and poll for

packets, **rac_poll()** is still useful since it will determine whether a reply is available for a specific <CLIENT, asynchronous handle> tuple.

rac_rcv()

rac_rcv() retrieves the results of a previous asynchronous RPC call, placing them in the buffer indicated in the **rac_send()** call and using the XDR decode function supplied there. It depends on the application to have ensured that a reply is present (using **rac_poll()**). If **rac_rcv()** is called before a reply has been received, it will block awaiting a reply.

All errors normally returned by the RPC client call functions may be returned here. In addition:

RPC_STALERACHANDLE Either the handle referred to by *h* is invalid or no call is currently in progress for the given <CLIENT, asynchronous handle> tuple.

Additionally, if a packet is present and its status is not **RPC_SUCCESS**, it is possible that the client credentials need refreshing. In this case, **RPC_AUTHERROR** is returned and the client should attempt to resend the call.

When a reply has been received, **rac_rcv()** will invoke the XDR decode procedure specified in the **rac_send()** call. After a call to **rac_rcv()**, the handle referred to by *h* is invalid. It may no longer be used in any asynchronous operation.

rac_send()

rac_send() initiates (sends to the server) an RPC call to the specified procedure. It does not await a reply from the server. *argsp* is the address of the procedure's arguments, *resultsp* is the address in which to place the results, *xargs* and *xresults* are XDR functions used to encode and decode respectively. Note: *resultsp* must be a valid pointer when **rac_rcv()** is called. *timeout* should contain the total amount of time the application is willing to wait for a reply.

Upon success, an opaque handle, known as the asynchronous handle, is returned. This handle is to be used in subsequent asynchronous calls to poll for the status of the

call (**rac_poll()**), receive the returned results of the call (**rac_rcv()**), or cancel the call (**rac_drop()**).

On failure, (void *) 0 is returned.

In case of failure, the application may retrieve the RPC failure code by calling **clnt_geterr()** immediately after a **rac_send()** failure (see **rpc(3N)**). Possible errors include both transient problems (such as transport failures) and permanent ones (such as XDR encoding failures).

Multiple **rac_send** s on the same client handle are permitted, but may introduce unpredictable perturbations to the current timeout and retry model used by the RPC library.

The interface imposes a limit on the amount of time a call may be in progress before it is considered to have failed. This method was chosen over limitations on the number of retries because of a desire for transport independence.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

poll(2) , **rpc(3N)** , **rpc_clnt_create(3N)** , **rpc_clnt_calls(3N)** , **xdr(3N)** , **attributes(5)**

WARNINGS

The RAC interface is not the recommended interface for having multiple RPC requests outstanding. The preferred method of accomplishing this in the Solaris environment is to use synchronous RPC calls with threads. The RAC interface is provided as a service to developers interested in porting RPC applications to Solaris 2.0. Use of this interface will degrade the performance of normal synchronous RPC calls (see **rpc_clnt_calls(3N)**). For these reasons, use of this interface is disparaged.

The library **librac** must be linked before **libns1** to use RAC. If the libraries are not linked in the correct order, then the results are indeterminate.

NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

| | |
|------------------------|---|
| NAME | rpcsec_gss – security flavor incorporating |
| SYNOPSIS | cc [<i>flag...</i>] <i>file...</i> -l <i>library...</i> -I#include <rpc/rpcsec_gss.h> (void); |
| DESCRIPTION | <p>RPCSEC_GSS is a security flavor which sits "on top" of the GSS-API (Generic Security Service API) for network transmissions. Applications using RPCSEC_GSS can take advantage of GSS-API security features; moreover, they can use any security mechanism (such as RSA public key or Kerberos) that works with the GSS-API.</p> <p>The GSS-API offers two security services beyond the traditional authentication services (AUTH_DES, AUTH_SYS, and AUTH_KERB): integrity and privacy. With integrity, the system uses cryptographic checksumming to ensure the authenticity of a message (authenticity of originator, recipient, and data); privacy provides additional security by encrypting data. Applications using RPCSEC_GSS specify which service they wish to use. Type of security service is mechanism-independent.</p> <p>Before exchanging data with a peer, an application must establish a context for the exchange. RPCSEC_GSS provides a single function for this purpose, <code>rpc_gss_seccreate()</code>, which allows the application to specify the security mechanism, Quality of Protection (QOP), and type of service at context creation. (The QOP parameter sets the cryptographic algorithms to be used with integrity or privacy, and is mechanism-dependent.) Once a context is established, applications can reset the QOP and type of service for each data unit exchanged, if desired.</p> <p>Valid mechanisms and QOPs may be obtained from configuration files or from the name service. Each mechanism has a default QOP.</p> <p>Contexts are destroyed with the usual RPC <code>auth_destroy()</code> call.</p> |
| Data Structures | <p>Some of the data structures used by the RPCSEC_GSS package are shown below.</p> <pre>rpc_gss_service_t</pre> <p>This enum defines the types of security services the context may have. <code>rpc_gss_seccreate()</code> takes this as one argument when setting the service type for a session.</p> <pre>typedef enum { rpc_gss_svc_default = 0, rpc_gss_svc_none = 1, rpc_gss_svc_integrity = 2, rpc_gss_svc_privacy = 3 } rpc_gss_service_t ;</pre> |

```
rpc_gss_options_req_t
```

Structure containing options passed directly through to the GSS-API. **rpc_gss_seccreate()** takes this as an argument when creating a context. (Because the GSS-API is not currently exposed, this data type is mentioned for informational purposes only. The programmer should set this to NULL.)

```
typedef struct {
    int      req_flags; GSS request bits
    int      time_req; requested credential lifetime
    gss_cred_id_t  my_cred; GSS credential struct
    gss_channel_bindings_t;
    input_channel_bindings;
} rpc_gss_options_req_t ;
```

```
rpc_gss_OID
```

This data type is used by in-kernel RPC routines, and thus is mentioned here for informational purposes only.

```
typedef struct {
    u_int length;
    void *elements
} *rpc_gss_OID;
```

```
rpc_gss_options_ret_t
```

Structure containing GSS-API options returned to the calling function, **rpc_gss_seccreate()**. **MAX_GSS_MECH** is defined as 128. (Because the GSS-API is not currently exposed, this data type is mentioned for informational purposes only. Set this to NULL in order to use default values.)

```
typedef struct {
    int major_status;
    int minor_status;
    u_int rpcsec_version vers. of RPCSEC_GSS
    int ret_flags
    int time_req
    gss_ctx_id_t gss_context;
    char actual_mechanism[MAX_GSS_MECH]; mechanism used
} rpc_gss_options_ret_t;
```

```
rpc_gss_principal_t
```

The (mechanism-dependent, opaque) client principal type. Used as an argument to the **rpc_gss_get_principal_name()** function, and in the `gsscred` table. Also referenced by the `rpc_gss_rawcred_t` structure for raw credentials (see below).

```
typedef struct {
    int len;
    char name[1];
} *rpc_gss_principal_t;
```

`rpc_gss_rawcred_t`

Structure for raw credentials. Used by **rpc_gss_getcred()** and **rpc_gss_set_callback()**.

```
typedef struct {
    u_int version; RPC version #
    char *mechanism; security mechanism
    char *qop; Quality of Protection
    rpc_gss_principal_t client_principal; client name
    char *svc_principal; server name
    rpc_gss_service_t service; service (integrity, etc.)
} rpc_gss_rawcred_t;
```

`rpc_gss_ucred_t`

Structure for UNIX credentials. Used by **rpc_gss_getcred()** as an alternative to `rpc_gss_rawcred_t`.

```
typedef struct {
    uid_t uid; user ID
    gid_t gid; group ID
    short gidlen;
    git_t *gidlist; list of groups
} rpc_gss_ucred_t;
```

`rpc_gss_callback_t`

Callback structure used by **rpc_gss_set_callback()**.

```
typedef struct {
    u_int program;      RPC program #
    u_int version;     RPC version #
    bool_t (*callback)(); user-defined callback routine
} rpc_gss_callback_t;
```

rpc_gss_lock_t

Structure used by a callback routine to enforce a particular QOP and service for a session. The `locked` field is normally set to `FALSE`; the server sets it to `TRUE` in order to lock the session. (A locked context will reject all requests having different QOP and service values than those found in the `raw_cred` structure.) For more information, see the `rpc_gss_set_callback(3N)` man page.

```
typedef struct {
    bool_t locked;
    rpc_gss_rawcred_t *raw_cred;
} rpc_gss_lock_t;
```

rpc_gss_error_t

Structure used by `rpc_gss_get_error()` to fetch an error code when a `RPCSEC_GSS` routine fails.

```
typedef struct {
    int rpc_gss_error;
    int system_error; same as errno
} rpc_gss_error_t;
```

Index to Routines

The following lists `RPCSEC_GSS` routines and the manual reference pages on which they are described. An (S) indicates it is a server-side function:

| Routine (Manual Page) | Description |
|--|---|
| <code>rpc_gss_seccreate(3N)</code> | Create a secure <code>RPCSEC_GSS</code> context |
| <code>rpc_gss_set_defaults(3N)</code> | Switch service, QOP for a session |
| <code>rpc_gss_max_data_length(3N)</code> | Get maximum data length allowed by transport |
| <code>rpc_gss_set_svc_name(3N)</code> | Set server's principal name (S) |

- rpc_gss_getcred(3N)** Get credentials of caller (S)
- rpc_gss_set_callback(3N)** Specify callback to see context use (S)
- rpc_gss_get_principal_name(3N)** Get client principal name (S)
- rpc_gss_svc_max_data_length(3N)** Get maximum data length allowed by transport (S)
- rpc_gss_get_error(3N)** Get error number
- rpc_gss_get_mechanisms(3N)** Get valid mechanism strings
- rpc_gss_get_mech_info(3N)** Get valid QOP strings, current service
- rpc_gss_get_versions(3N)** Get supported RPCSEC_GSS versions
- rpc_gss_is_installed(3N)** Checks if a mechanism is installed
- rpc_gss_mech_to_oid(3N)** Maps ASCII mechanism to OID representation
- rpc_gss_qop_to_num(3N)** Maps ASCII QOP, mechanism to u_int number

Utilities The `gsscred` utility manages the `gsscred` table, which contains mappings of principal names between network and local credentials. See `gsscred(1M)`.

- FILES**
- `/etc/gss/mech` List of installed mechanisms
 - `/etc/gss/qop` List of valid QOPs

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------|
| MT-Level | MT-Safe |
| Packages | SUNWrsg, SUNWrsgx |

SEE ALSO

`gsscred(1M)`, `rpc(3N)`, `rpc_clnt_auth(3N)`, `xdr(3N)`, `attributes(5)`,
`environ(5)`

ONC+ Developer's Guide

Network Working Group RFC 2078

| | |
|-------------------------|---|
| NAME | rpc_soc, authdes_create, authunix_create, authunix_create_default, callrpc, clnt_broadcast, clntraw_create, clnttcp_create, clntudp_bufcreate, clntudp_create, get_myaddress, getrpcport, pmap_getmaps, pmap_getport, pmap_rmtcall, pmap_set, pmap_unset, registerrpc, svc_fds, svc_getcaller, svc_getreq, svc_register, svc_unregister, svctcp_create, svcfd_create, svcraw_create, svctcp_create, svcudp_bufcreate, svcudp_create, xdr_authunix_parms - obsolete library routines for RPC |
| DESCRIPTION | <p>RPC routines allow C programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply. Finally, the procedure call returns to the client.</p> <p>The routines described in this manual page have been superseded by other routines. The preferred routine is given after the description of the routine. New programs should use the preferred routines, as support for the older interfaces may be dropped in future releases.</p> |
| File Descriptors | <p>Transport independent RPC uses TLI as its transport interface instead of sockets.</p> <p>Some of the routines described in this section (such as clnttcp_create()) take a pointer to a file descriptor as one of the parameters. If the user wants the file descriptor to be a socket, then the application will have to be linked with both <code>librpcsoc</code> and <code>libnsl</code>. If the user passed <code>RPC_ANYSOCK</code> as the file descriptor, and the application is linked with <code>libnsl</code> only, then the routine will return a TLI file descriptor and not a socket.</p> |
| Routines | <p>The following routines require that the header <code><rpc/rpc.h></code> be included. The symbol <code>PORTMAP</code> should be defined so that the appropriate function declarations for the old interfaces are included through the header files.</p> <pre>#define PORTMAP #include <rpc/rpc.h></pre> <p>AUTH <i>*authdes_create(char * name , uint_t window , struct sockaddr_in * syncaddr , des_block * ckey);</i></p> <p>authdes_create() is the first of two routines which interface to the RPC secure authentication system, known as DES authentication. The second is authdes_getucred() , below. Note: the keyserver daemon keyerv(1M) must be running for the DES authentication system to work.</p> <p>authdes_create() , used on the client side, returns an authentication handle that will enable the use of the secure authentication system. The first</p> |

parameter *name* is the network name, or *netname*, of the owner of the server process. This field usually represents a hostname derived from the utility routine **host2netname()**, but could also represent a user name using **user2netname()** (see **secure_rpc(3N)**). The second field is window on the validity of the client credential, given in seconds. A small window is more secure than a large one, but choosing too small of a window will increase the frequency of resynchronizations because of clock drift. The third parameter *syncaddr* is optional. If it is NULL, then the authentication system will assume that the local clock is always in sync with the server's clock, and will not attempt resynchronizations. If an address is supplied, however, then the system will use the address for consulting the remote time service whenever resynchronization is required. This parameter is usually the address of the RPC server itself. The final parameter *key* is also optional. If it is NULL, then the authentication system will generate a random DES key to be used for the encryption of credentials. If it is supplied, however, then it will be used instead.

Warning: this routine exists for backward compatibility only, and is obsolete by **authdes_seccreate()** (see **secure_rpc(3N)**).

AUTH *authunix_create(char * host , uid_t uid , gid_t gid , int grouplen , gid_t * gidlist) ;

Create and return an RPC authentication handle that contains .UX authentication information. The parameter *host* is the name of the machine on which the information was created; *uid* is the user's user ID; *gid* is the user's current group ID; *grouplen* and *gidlistp* refer to a counted array of groups to which the user belongs.

Warning: it is not very difficult to impersonate a user.

Warning: this routine exists for backward compatibility only, and is obsolete by **authsys_create()** (see **rpc_clnt_auth(3N)**).

AUTH *authunix_create_default(void)

Call **authunix_create()** with the appropriate parameters.

Warning: this routine exists for backward compatibility only, and is obsolete by **authsys_create_default()** (see **rpc_clnt_auth(3N)**).

callrpc(char * host , rpcprog_t prognum , rpcvers_t versnum , rpcproc_t procnum , xdrproc_t inproc , char * in , xdrproc_t outproc , char * out);

Call the remote procedure associated with *prognum*, *versnum*, and *procnum* on the machine, *host*. The parameter *inproc* is used to encode the

procedure's parameters, and *outproc* is used to decode the procedure's results; *in* is the address of the procedure's argument, and *out* is the address of where to place the result(s). This routine returns 0 if it succeeds, or the value of `enum clnt_stat` cast to an integer if it fails. The routine `clnt_permpno()` (see `rpc_clnt_calls(3N)`) is handy for translating failure statuses into messages.

Warning: you do not have control of timeouts or authentication using this routine. This routine exists for backward compatibility only, and is obsoleted by `rpc_call()` (see `rpc_clnt_calls(3N)`).

```
enum clnt_stat clnt_broadcast(rpcprog_t prognum , rpcvers_t versnum ,
rpcproc_t procnum , xdrproc_t inproc , char * in , xdrproc_t outproc , char * out , resultproc_t eachresult );
```

Like `callrpc()` , except the call message is broadcast to all locally connected broadcast nets. Each time the caller receives a response, this routine calls `eachresult()` , whose form is:

```
eachresult(char * out , struct sockaddr_in * addr );
```

where *out* is the same as *out* passed to `clnt_broadcast()` , except that the remote procedure's output is decoded there; *addr* points to the address of the machine that sent the results. If `eachresult()` returns 0 `clnt_broadcast()` waits for more replies; otherwise it returns with appropriate status. If `eachresult()` is NULL, `clnt_broadcast()` returns without waiting for any replies.

Warning: broadcast packets are limited in size to the maximum transfer unit of the transports involved. For Ethernet, the callers argument size is approximately 1500 bytes. Since the call message is sent to all connected networks, it may potentially lead to broadcast storms. `clnt_broadcast()` uses SB AUTH_SYS credentials by default (see `rpc_clnt_auth(3N)`).

Warning: this routine exists for backward compatibility only, and is obsoleted by `rpc_broadcast()` (see `rpc_clnt_calls(3N)`).

```
CLIENT *clntrow_create(rpcprog_t prognum , rpcvers_t versnum );
```

This routine creates an internal, memory-based RPC client for the remote program *prognum* , version *versnum* . The transport used to pass messages to the service is actually a buffer within the process's address space, so the corresponding RPC server should live in the same address space; see `svccraw_create()` . This allows simulation of RPC and acquisition of RPC overheads, such as round trip times, without any kernel interference. This routine returns NULL if it fails.

Warning: this routine exists for backward compatibility only, and has the same functionality as `clnt_raw_create()` (see `rpc_clnt_create(3N)`), which obsoletes it.

```
CLIENT *clnttcp_create(struct sockaddr_in * addr , rpcprog_t prognum ,
rpcvers_t versnum , int * fdp , uint_t sendsz , uint_t recvsz );
```

This routine creates an RPC client for the remote program *prognum*, version *versnum*; the client uses TCP/IP as a transport. The remote program is located at Internet address *addr*. If *addr* ⇒ *sin_port* is 0, then it is set to the actual port that the remote program is listening on (the remote `rpcbind` service is consulted for this information). The parameter **fdp* is a file descriptor, which may be open and bound; if it is `RPC_ANYSOCK`, then this routine opens a new one and sets **fdp*. Refer to the File Descriptor section for more information. Since TCP-based RPC uses buffered I/O, the user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails.

Warning: this routine exists for backward compatibility only. `clnt_create()`, `clnt_tli_create()`, or `clnt_vc_create()` (see `rpc_clnt_create(3N)`) should be used instead.

```
CLIENT *clntudp_bufcreate(struct sockaddr_in * addr , rpcprog_t
prognum , rpcvers_t versnum , struct timeval wait , int * fdp , uint_t sendsz
, uint_t recvsz );
```

Create a client handle for the remote program *prognum*, on *versnum*; the client uses UDP/IP as the transport. The remote program is located at the Internet address *addr*. If *addr* ⇒ *sin_port* is 0, it is set to port on which the remote program is listening on (the remote `rpcbind` service is consulted for this information). The parameter **fdp* is a file descriptor, which may be open and bound; if it is `RPC_ANYSOCK`, then this routine opens a new one and sets **fdp*. Refer to the File Descriptor section for more information. The UDP transport resends the call message in intervals of *wait* time until a response is received or until the call times out. The total time for the call to time out is specified by `clnt_call()` (see `rpc_clnt_calls(3N)`). If successful it returns a client handle, otherwise it returns NULL. The error can be printed using the `clnt_pcreateerror()` (see `rpc_clnt_create(3N)`) routine.

The user can specify the maximum packet size for sending and receiving by using *sendsz* and *recvsz* arguments for UDP-based RPC messages.

Warning: if *addr* ⇒ *sin_port* is 0 and the requested version number *versnum* is not registered with the remote portmap service, it returns a handle if at least

a version number for the given program number is registered. The version mismatch is discovered by a `clnt_call()` later (see `rpc_clnt_calls(3N)`).

Warning: this routine exists for backward compatibility only.

`clnt_tli_create()` or `clnt_dg_create()` (see `rpc_clnt_create(3N)`) should be used instead.

```
CLIENT *clntudp_create(struct sockaddr_in * addr , rpcprog_t prognum ,
rpcvers_t versnum , struct timeval wait , int * fdp );
```

This routine creates an RPC client handle for the remote program *prognum* , version *versnum* ; the client uses UDP/IP as a transport. The remote program is located at Internet address *addr* . If *addr* \Rightarrow *sin_port* is 0 , then it is set to actual port that the remote program is listening on (the remote `rpcbind` service is consulted for this information). The parameter **fdp* is a file descriptor, which may be open and bound; if it is `RPC_ANYSOCK` , then this routine opens a new one and sets **fdp* . Refer to the File Descriptor section for more information. The UDP transport resends the call message in intervals of *wait* time until a response is received or until the call times out. The total time for the call to time out is specified by `clnt_call()` (see `rpc_clnt_calls(3N)`). `clntudp_create()` returns a client handle on success, otherwise it returns NULL. The error can be printed using the `clnt_pcreateerror()` (see `rpc_clnt_create(3N)`) routine.

Warning: since UDP-based RPC messages can only hold up to 8 Kbytes of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

Warning: this routine exists for backward compatibility only. `clnt_create()` , `clnt_tli_create()` , or `clnt_dg_create()` (see `rpc_clnt_create(3N)`) should be used instead.

```
void get_myaddress(struct sockaddr_in * addr );
```

Places the local system's IP address into **addr* , without consulting the library routines that deal with `/etc/hosts` . The port number is always set to `htons(PMAPPORT)` .

Warning: this routine is only intended for use with the RPC library. It returns the local system's address in a form compatible with the RPC library, and should not be taken as the system's actual IP address. In fact, the **addr* buffer's host address part is actually zeroed. This address may have only local significance and should NOT be assumed to be an address that can be used to connect to the local system by remote systems or processes.

Warning: this routine remains for backward compatibility only. The routine `netdir_getbyname()` (see `netdir(3N)`) should be used with the name

HOST_SELF to retrieve the local system's network address as a *netbuf* structure.

```
u_short getrpcport(char * host , rpcprog_t prognum , rpcvers_t versnum ,
rpcprot_t proto );
```

getrpcport() returns the port number for the version *versnum* of the RPC program *prognum* running on *host* and using protocol *proto* . **getrpcport()** returns 0 if the RPC system failed to contact the remote portmap service, the program associated with *prognum* is not registered, or there is no mapping between the program and a port.

Warning: This routine exists for backward compatibility only. Enhanced functionality is provided by **rpcb_getaddr()** (see **rpcbind(3N)**).

```
struct pmaplist *pmap_getmaps(struct sockaddr_in * addr );
```

A user interface to the `portmap` service, which returns a list of the current RPC program-to-port mappings on the host located at IP address *addr* . This routine can return NULL . The command `'rpcinfo -p'` uses this routine.

Warning: this routine exists for backward compatibility only, enhanced functionality is provided by **rpcb_getmaps()** (see **rpcbind(3N)**).

```
u_short pmap_getport(struct sockaddr_in * addr , rpcprog_t prognum ,
rpcvers_t versnum , rpcprot_t protocol );
```

A user interface to the `portmap` service, which returns the port number on which waits a service that supports program *prognum* , version *versnum* , and speaks the transport protocol associated with *protocol* . The value of *protocol* is most likely IPPROTO_UDP or IPPROTO_TCP . A return value of 0 means that the mapping does not exist or that the RPC system failed to contact the remote `portmap` service. In the latter case, the global variable `rpc_createerr` contains the RPC status.

Warning: this routine exists for backward compatibility only, enhanced functionality is provided by **rpcb_getaddr()** (see **rpcbind(3N)**).

```
enum clnt_stat pmap_rmtcall(struct sockaddr_in * addr , rpcprog_t prognum
, rpcvers_t versnum , rpcproc_t procnum , caddr_t in , xdrproct_t inproc ,
caddr_t out , xdrproct_t outproc , struct timeval tout , rpcport_t * portp );
```

Request that the `portmap` on the host at IP address **addr* make an RPC on the behalf of the caller to a procedure on that host. **portp* is modified to the program's port number if the procedure succeeds. The definitions of other

parameters are discussed in `callrpc()` and `clnt_call()` (see `rpc_clnt_calls(3N)`).

Note: this procedure is only available for the UDP transport.

Warning: if the requested remote procedure is not registered with the remote `portmap` then no error response is returned and the call times out. Also, no authentication is done.

Warning: this routine exists for backward compatibility only, enhanced functionality is provided by `rpcb_rmtcall()` (see `rpcbind(3N)`).

bool_t pmmap_set(rpcprog_t prognum , rpcvers_t versnum , rpcprot_t protocol , u_short port);

A user interface to the `portmap` service, that establishes a mapping between the triple [`prognum` , `versnum` , `protocol`] and `port` on the machine's `portmap` service. The value of `protocol` may be `IPPROTO_UDP` or `IPPROTO_TCP`. Formerly, the routine failed if the requested `port` was found to be in use. Now, the routine only fails if it finds that `port` is still bound. If `port` is not bound, the routine completes the requested registration. This routine returns 1 if it succeeds, 0 otherwise. Automatically done by `svc_register()`.

Warning: this routine exists for backward compatibility only, enhanced functionality is provided by `rpcb_set()` (see `rpcbind(3N)`).

bool_t pmmap_unset(rpcprog_t prognum , rpcvers_t versnum);

A user interface to the `portmap` service, which destroys all mapping between the triple [`prognum` , `versnum` , `all-protocols`] and `port` on the machine's `portmap` service. This routine returns one if it succeeds, 0 otherwise.

Warning: this routine exists for backward compatibility only, enhanced functionality is provided by `rpcb_unset()` (see `rpcbind(3N)`).

int svc_fds;

A global variable reflecting the RPC service side's read file descriptor bit mask; it is suitable as a parameter to the `select()` call. This is only of interest if a service implementor does not call `svc_run()`, but rather does his own asynchronous event processing. This variable is read-only (do not pass its address to `select()`!), yet it may change after calls to `svc_getreq()` or any creation routines. Similar to `svc_fdset`, but limited to 32 descriptors.

Warning: this interface is obsolete by `svc_fdset` (see `rpc_svc_calls(3N)`).

```
struct sockaddr_in * svc_getcaller(SVCXPRT * xprt );
```

This routine returns the network address, represented as a `struct sockaddr_in`, of the caller of a procedure associated with the RPC service transport handle, `xprt`.

Warning: this routine exists for backward compatibility only, and is obsolete. The preferred interface is `svc_getrpccaller()` (see `rpc_svc_reg(3N)`), which returns the address as a `struct netbuf`.

```
void svc_getreq(int rdfs );
```

This routine is only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. It is called when the `select()` call has determined that an RPC request has arrived on some RPC file descriptors; `rdfs` is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of `rdfs` have been serviced. This routine is similar to `svc_getreqset()` but is limited to 32 descriptors.

Warning: this interface is obsoleted by `svc_getreqset()`.

```
SVCXPRT *svdfd_create(int fd , uint_t sendsz , uint_t recvsz );
```

Create a service on top of any open and bound descriptor. Typically, this descriptor is a connected file descriptor for a stream protocol. Refer to the `File Descriptor` section for more information. `sendsz` and `recvsz` indicate sizes for the send and receive buffers. If they are 0, a reasonable default is chosen.

Warning: this interface is obsoleted by `svc_fd_create()` (see `rpc_svc_create(3N)`).

```
SVCXPRT *svcrow_create(void);
```

This routine creates an internal, memory-based RPC service transport, to which it returns a pointer. The transport is really a buffer within the process's address space, so the corresponding RPC client should live in the same address space; see `clntraw_create()`. This routine allows simulation of RPC and acquisition of RPC overheads (such as round trip times), without any kernel interference. This routine returns NULL if it fails.

Warning: this routine exists for backward compatibility only, and has the same functionality of `svc_raw_create()` (see `rpc_svc_create(3N)`), which obsoletes it.

SVCXPRT *svctcp_create(int *fd* , uint_t *sendsz* , uint_t *recvsz*) ;

This routine creates a TCP/IP-based RPC service transport, to which it returns a pointer. The transport is associated with the file descriptor *fd* , which may be `RPC_ANYSOCK` , in which case a new file descriptor is created. If the file descriptor is not bound to a local TCP port, then this routine binds it to an arbitrary port. Refer to the `File Descriptor` section for more information. Upon completion, *xprt* \Rightarrow *xp_fd* is the transport's file descriptor, and *xprt* \Rightarrow *xp_port* is the transport's port number. This routine returns NULL if it fails. Since TCP-based RPC uses buffered I/O, users may specify the size of buffers; values of 0 choose suitable defaults.

Warning: this routine exists for backward compatibility only. `svc_create()` , `svc_tli_create()` , or `svc_vc_create()` (see `rpc_svc_create(3N)`) should be used instead.

SVCXPRT *svcudp_bufcreate(int *fd* , uint_t *sendsz* , uint_t *recvsz*);

This routine creates a UDP/IP-based RPC service transport, to which it returns a pointer. The transport is associated with the file descriptor *fd* . If *fd* is `RPC_ANYSOCK` , then a new file descriptor is created. If the file descriptor is not bound to a local UDP port, then this routine binds it to an arbitrary port. Upon completion, *xprt* \Rightarrow *xp_fd* is the transport's file descriptor, and *xprt* \Rightarrow *xp_port* is the transport's port number. Refer to the `File Descriptor` section for more information. This routine returns NULL if it fails.

The user specifies the maximum packet size for sending and receiving UDP-based RPC messages by using the *sendsz* and *recvsz* parameters.

Warning: this routine exists for backward compatibility only. `svc_tli_create()` , or `svc_dg_create()` (see `rpc_svc_create(3N)`) should be used instead.

SVCXPRT *svcudp_create(int *fd*);

This routine creates a UDP/IP-based RPC service transport, to which it returns a pointer. The transport is associated with the file descriptor *fd* , which may be `RPC_ANYSOCK` , in which case a new file descriptor is created. If the file descriptor is not bound to a local UDP port, then this routine binds it to an arbitrary port. Upon completion, *xprt* \Rightarrow *xp_fd* is the transport's file descriptor, and *xprt* \Rightarrow *xp_port* is the transport's port number. This routine returns NULL if it fails.

Warning: since UDP-based RPC messages can only hold up to 8 Kbytes of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

Warning: this routine exists for backward compatibility only. `svc_create()`, `svc_tli_create()`, or `svc_dg_create()` (see `rpc_svc_create(3N)`) should be used instead.

```
registerrpc(rpcprog_t prognum, rpcvers_t versnum, rpcproc_t procnum,
char *(procname)(), xdrproc_t inproc, xdrproc_t outproc);
```

Register program *prognum*, procedure *procname*, and version *versnum* with the RPC service package. If a request arrives for program *prognum*, version *versnum*, and procedure *procnum*, *procname* is called with a pointer to its parameter(s); *procname* should return a pointer to its static result(s); *inproc* is used to decode the parameters while *outproc* is used to encode the results. This routine returns 0 if the registration succeeded, -1 otherwise.

`svc_run()` must be called after all the services are registered.

Warning: this routine exists for backward compatibility only, and is obsoleted by `rpc_reg()`.

```
bool_t svc_register(SVCXPRT * xprt, rpcprog_t prognum, rpcvers_t versnum,
void *(dispatch)(), int protocol);
```

Associates *prognum* and *versnum* with the service dispatch procedure, *dispatch*. If *protocol* is 0, the service is not registered with the `portmap` service. If *protocol* is non-zero, then a mapping of the triple [*prognum*, *versnum*, *protocol*] to *xprt* \Rightarrow *xp_port* is established with the local `portmap` service (generally *protocol* is 0, `IPPROTO_UDP` or `IPPROTO_TCP`). The procedure *dispatch* has the following form:

```
dispatch(struct svc_req * request, SVCXPRT * xprt);
```

The `svc_register()` routine returns one if it succeeds, and 0 otherwise.

Warning: this routine exists for backward compatibility only; enhanced functionality is provided by `svc_reg()`.

```
void svc_unregister(rpcprog_t prognum, rpcvers_t versnum);
```

Remove all mapping of the double [*prognum*, *versnum*] to dispatch routines, and of the triple [*prognum*, *versnum*, *all-protocols*] to port number from `portmap`.

Warning: this routine exists for backward compatibility, enhanced functionality is provided by `svc_unreg()` .

bool_t xdr_authunix_parms(XDR * xdrs , struct authunix_parms * aupp) ;

Used for describing UNIX credentials. This routine is useful for users who wish to generate these credentials without using the RPC authentication package.

Warning: this routine exists for backward compatibility only, and is obsoleted by `xdr_authsys_parms()` (see `rpc_xdr(3N)`).

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

`keyserv(1M)` , `rpcbind(1M)` , `rpcinfo(1M)` , `netdir(3N)` , `netdir_getbyname(3N)` , `rpc(3N)` , `rpc_clnt_auth(3N)` , `rpc_clnt_calls(3N)` , `rpc_clnt_create(3N)` , `rpc_svc_calls(3N)` , `rpc_svc_create(3N)` , `rpc_svc_err(3N)` , `rpc_svc_reg(3N)` , `rpc_xdr(3N)` , `rpcbind(3N)` , `secure_rpc(3N)` , `select(3C)` , `xdr_authsys_parms(3N)` , `libnsl(4)` , `librpcsoc(4)` , `attributes(5)`

NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

NAME | `rpc_svc_calls`, `svc_dg_enablecache`, `svc_done`, `svc_exit`, `svc_fdset`, `svc_freeargs`, `svc_getargs`, `svc_getreq_common`, `svc_getreq_poll`, `svc_getreqset`, `svc_getrpccaller`, `svc_max_pollfd`, `svc_pollfd`, `svc_run`, `svc_sendreply` – library routines for RPC servers

DESCRIPTION | These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.

These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as `svc_run()`) are called when the server is initiated.

In the current implementation, the service transport handle `SVCXPRT` contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. However, when a server is operating in the Automatic or User MT modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.

Routines

```
#include <rpc/rpc.h>
```

```
int svc_dg_enablecache(SVCXPRT * xprt , const uint_t cache_size );
```

This function allocates a duplicate request cache for the service endpoint `xprt`, large enough to hold `cache_size` entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.

This function is safe in MT applications.

```
int svc_done(SVCXPRT * xprt );
```

This function frees resources allocated to service a client request directed to the service endpoint `xprt`. This call pertains only to servers executing in the User MT mode. In the User MT mode, service procedures must invoke this call before returning, either after a client request has been serviced, or after an error or abnormal condition that prevents a reply from being sent. After `svc_done()` is invoked, the service endpoint `xprt` should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the `rpc_control()` call.

This function is safe in MT applications. It will have no effect if invoked in modes other than the User MT mode.

void svc_exit(void);

This function when called by any of the RPC server procedure or otherwise, destroys all services registered by the server and causes **svc_run()** to return.

If RPC server activity is to be resumed, services must be reregistered with the RPC library either through one of the **rpc_svc_create(3N)** functions, or using **xprt_register(3N)**.

svc_exit() has global scope and ends all RPC server activity.

fd_set svc_fdset;

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest if service implementors do not call **svc_run()**, but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to **svc_getregset()** or any creation routines. Do not pass its address to **select(3C)**! Instead, pass the address of a copy.

MT applications executing in either the Automatic MT mode or the user MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

svc_fdset is limited to 1024 file descriptors and is considered obsolete. Use of **svc_pollfd** is recommended instead.

pollfd_t *svc_pollfd;

A global variable pointing to an array of **pollfd_t** structures reflecting the RPC server's read file descriptor array. This is only of interest if service implementors do not call **svc_run()** but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to **svc_getreg_poll()** or any creation routines. Do not pass its address to **poll(2)**! Instead, pass the address of a copy.

By default, **svc_pollfd** is limited to 1024 entries. Use **rpc_control(3N)** to remove this limitation.

MT applications executing in either the Automatic MT mode or the user MT mode should never be read this variable. They should use auxiliary threads to do asynchronous event processing.

int svc_max_pollfd;

A global variable containing the maximum length of the *svc_pollfd* array. This variable is read-only, and it may change after calls to *svc_getreg_poll()* or any creation routines.

```
bool_t svc_freeargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in );
```

A function macro that frees any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using *svc_getargs()*. This routine returns *TRUE* if the results were successfully freed, and *FALSE* otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

```
bool_t svc_getargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in );
```

A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xprt*. The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns *TRUE* if decoding succeeds, and *FALSE* otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

```
void svc_getreq_common(const int fd );
```

This routine is called to handle a request on the given file descriptor.

```
void svc_getreq_poll(struct pollfd * pfdp , const int pollretval );
```

This routine is only of interest if a service implementor does not call *svc_run()*, but instead implements custom asynchronous event processing. It is called when *poll(2)* has determined that an RPC request has arrived on some RPC file descriptors; *pollretval* is the return value from *poll(2)* and *pfdp* is the array of *pollfd* structures on which the *poll(2)* was done. It is assumed to be an array large enough to contain the maximal number of descriptors allowed.

This function macro is unsafe in MT applications.

```
void svc_getreqset(fd_set * rdfs );
```

This routine is only of interest if a service implementor does not call **svc_run()** , but instead implements custom asynchronous event processing. It is called when **select(3C)** has determined that an RPC request has arrived on some RPC file descriptors; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfds* have been serviced.

This function macro is unsafe in MT applications.

struct netbuf *svc_getrpccaller(const SVCXPRT * *xprt*);

The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xprt* .

This function macro is safe in MT applications.

void svc_run(void);

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using **svc_getreq_poll()** when one arrives. This procedure is usually waiting for the **poll(2)** library call to return.

Applications executing in the Automatic or User MT modes should invoke this function exactly once. In the Automatic MT mode, it will create threads to service client requests. In the User MT mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

bool_t svc_sendreply(const SVCXPRT * *xprt* , const xdrproc_t *outproc* , const caddr_t *out*);

Called by an RPC service's dispatch routine to send the results of a remote procedure call. The parameter *xprt* is the request's associated transport handle; *outproc* is the XDR routine which is used to encode the results; and *out* is the address of the results. This routine returns **TRUE** if it succeeds, **FALSE** otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|------------------|
| MT-Level | See NOTES below. |

SEE ALSO

`rpcgen(1)` , `poll(2)` , `rpc(3N)` , `rpc_control(3N)` ,
`rpc_svc_create(3N)` , `rpc_svc_err(3N)` , `rpc_svc_reg(3N)` ,
`select(3C)` , `xprt_register(3N)` , `attributes(5)`

NOTES

`svc_dg_enablecache()` and `svc_getrpccaller()` are safe in multithreaded applications. `svc_freeargs()` , `svc_getargs()` , and `svc_sendreply()` are safe in MT applications utilizing the Automatic or User MT modes. `svc_getreq_common()` , `svc_getreqset()` , and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

| | | | | | | | |
|--------------------|---|------------------|--|------------------|---|------------|---|
| NAME | rpc_svc_create, svc_control, svc_create, svc_destroy, svc_dg_create, svc_fd_create, svc_raw_create, svc_tli_create, svc_tp_create, svc_vc_create – library routines for the creation of server handles | | | | | | |
| DESCRIPTION | These routines are part of the RPC library which allows C language programs to make procedure calls on servers across the network. These routines deal with the creation of service handles. Once the handle is created, the server can be invoked by calling svc_run() . | | | | | | |
| Routines | See rpc(3N) for the definition of the SVCXPRT data structure. <pre>#include <rpc/rpc.h></pre> <p>bool_t svc_control(SVCXPRT * svc , const uint_t req , void * info);</p> <p>A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i> , their argument types, and what they do are:</p> <table border="0"> <tr> <td style="vertical-align: top;">SVCGET_VERSQUIET</td> <td>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an RPC_PROGVERSMISMATCH error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the SVCGET_VERSQUIET request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an RPC_PROGVERSMISMATCH error will be returned; 1 indicates that the out of range request will be silently ignored.</td> </tr> <tr> <td style="vertical-align: top;">SVCSET_VERSQUIET</td> <td>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an RPC_PROGVERSMISMATCH error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0 , indicating normal server behavior and an RPC_PROGVERSMISMATCH error will be returned, or 1 , indicating that the out of range request should be silently ignored.</td> </tr> <tr> <td style="vertical-align: top;">SVCGET_XID</td> <td>Returns the transaction ID of connection-oriented (vc) and connectionless</td> </tr> </table> | SVCGET_VERSQUIET | If a request is received for a program number served by this server but the version number is outside the range registered with the server, an RPC_PROGVERSMISMATCH error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the SVCGET_VERSQUIET request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an RPC_PROGVERSMISMATCH error will be returned; 1 indicates that the out of range request will be silently ignored. | SVCSET_VERSQUIET | If a request is received for a program number served by this server but the version number is outside the range registered with the server, an RPC_PROGVERSMISMATCH error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0 , indicating normal server behavior and an RPC_PROGVERSMISMATCH error will be returned, or 1 , indicating that the out of range request should be silently ignored. | SVCGET_XID | Returns the transaction ID of connection-oriented (vc) and connectionless |
| SVCGET_VERSQUIET | If a request is received for a program number served by this server but the version number is outside the range registered with the server, an RPC_PROGVERSMISMATCH error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the SVCGET_VERSQUIET request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an RPC_PROGVERSMISMATCH error will be returned; 1 indicates that the out of range request will be silently ignored. | | | | | | |
| SVCSET_VERSQUIET | If a request is received for a program number served by this server but the version number is outside the range registered with the server, an RPC_PROGVERSMISMATCH error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0 , indicating normal server behavior and an RPC_PROGVERSMISMATCH error will be returned, or 1 , indicating that the out of range request should be silently ignored. | | | | | | |
| SVCGET_XID | Returns the transaction ID of connection-oriented (vc) and connectionless | | | | | | |

(dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the SVCGET_XID request, * *info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type SVCGET_XID, **svc_control()** will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET_XID in **clnt_control()**. See **clnt_create(3N)**

```
int svc_create(const void (* dispatch )(const struct svc_req *, const SVCXPRT
*), const rpcprog_t prognum , const rpcvers_t versnum , const char * nettype );
```

svc_create() creates server handles for all the transports belonging to the class *nettype* .

nettype defines a class of transports which can be used for a particular application. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database. If *nettype* is NULL, it defaults to netpath .

svc_create() registers itself with the rpcbind service (see **rpcbind(1M)**). *dispatch* is called when there is a remote procedure call for the given *prognum* and *versnum* ; this requires calling **svc_run()** (see **svc_run()** in **rpc_svc_reg(3N)**). If **svc_create()** succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

```
void svc_destroy(SVCXPRT * xprt );
```

A function macro that destroys the RPC service handle *xprt* . Destruction usually involves deallocation of private data structures, including *xprt* itself. Use of *xprt* is undefined after calling this routine.

```
SVCXPRT *svc_dg_create(const int fildes , const uint_t sendsz , const uint_t
recvsz );
```

This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns `NULL` if it fails, and an error message is logged. `sendsz` and `recvsz` are parameters used to specify the size of the buffers. If they are 0, suitable defaults are chosen. The file descriptor `fildev` should be open and bound. The server is not registered with `rpcbind(1M)`.

Warning: since connectionless-based RPC messages can only hold limited amount of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

SVCXPRT *svc_fd_create(const int *fildev*, const uint_t *sendsz*, const uint_t *recvsz*);

This routine creates a service on top of an open and bound file descriptor, and returns the handle to it. Typically, this descriptor is a connected file descriptor for a connection-oriented transport. `sendsz` and `recvsz` indicate sizes for the send and receive buffers. If they are 0, reasonable defaults are chosen. This routine returns `NULL` if it fails, and an error message is logged.

SVCXPRT *svc_raw_create(void);

This routine creates an RPC service handle and returns a pointer to it. The transport is really a buffer within the process's address space, so the corresponding RPC client should live in the same address space; (see `clnt_raw_create()` in `rpc_clnt_create(3N)`). This routine allows simulation of RPC and acquisition of RPC overheads (such as round trip times), without any kernel and networking interference. This routine returns `NULL` if it fails, and an error message is logged.

Note: `svc_run()` should not be called when the raw interface is being used.

SVCXPRT *svc_tli_create(const int *fildev*, const struct netconfig * *netconf*, const struct t_bind * *bindaddr*, const uint_t *sendsz*, const uint_t *recvsz*);

This routine creates an RPC server handle, and returns a pointer to it. `fildev` is the file descriptor on which the service is listening. If `fildev` is `RPC_ANYFD`, it opens a file descriptor on the transport specified by `netconf`. If the file descriptor is unbound and `bindaddr` is non-null `fildev` is bound to the address specified by `bindaddr`, otherwise `fildev` is bound to a default address chosen by the transport. In the case where the default address is chosen, the number of outstanding connect requests is set to 8 for connection-oriented transports. The user may specify the size of the send and receive buffers with the parameters `sendsz` and `recvsz`; values of 0 choose suitable defaults.

This routine returns `NULL` if it fails, and an error message is logged. The server is not registered with the `rpcbind(1M)` service.

```
SVCXPRT *svc_tp_create(const void (* dispatch )(const struct svc_req *, const
SVCXPRT *), const rpcprog_t prognum , const rpcvers_t versnum , const
struct netconfig * netconf );
```

`svc_tp_create()` creates a server handle for the network specified by `netconf`, and registers itself with the `rpcbind` service. `dispatch` is called when there is a remote procedure call for the given `prognum` and `versnum`; this requires calling `svc_run()`. `svc_tp_create()` returns the service handle if it succeeds, otherwise a `NULL` is returned and an error message is logged.

```
SVCXPRT *svc_vc_create(const int fildes , const uint_t sendsz , const uint_t
rcvsz );
```

This routine creates a connection-oriented RPC service and returns a pointer to it. This routine returns `NULL` if it fails, and an error message is logged. The users may specify the size of the send and receive buffers with the parameters `sendsz` and `rcvsz`; values of 0 choose suitable defaults. The file descriptor `fildes` should be open and bound. The server is not registered with the `rpcbind(1M)` service.

ATTRIBUTES

See `attributes` (5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`rpcbind(1M)`, `rpc(3N)`, `rpc_clnt_create(3N)`, `rpc_svc_calls(3N)`, `rpc_svc_err(3N)`, `rpc_svc_reg(3N)`, `attributes(5)`

| | |
|--------------------|---|
| NAME | rpc_svc_err, svcerr_auth, svcerr_decode, svcerr_noproc, svcerr_noprogram, svcerr_progvers, svcerr_systemerr, svcerr_weakauth – library routines for server side remote procedure call errors |
| DESCRIPTION | <p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines can be called by the server side dispatch function if there is any error in the transaction with the client.</p> |
| Routines | <p>See rpc(3N) for the definition of the SVCXPRT data structure.</p> <pre>#include <rpc/rpc.h></pre> <p>void svcerr_auth(const SVCXPRT * xprt , const enum auth_stat why);</p> <p>Called by a service dispatch routine that refuses to perform a remote procedure call due to an authentication error.</p> <p>void svcerr_decode(const SVCXPRT * xprt);</p> <p>Called by a service dispatch routine that cannot successfully decode the remote parameters (see svc_getargs() in rpc_svc_reg(3N)).</p> <p>void svcerr_noproc(const SVCXPRT * xprt);</p> <p>Called by a service dispatch routine that does not implement the procedure number that the caller requests.</p> <p>void svcerr_noprogram(const SVCXPRT * xprt);</p> <p>Called when the desired program is not registered with the RPC package. Service implementors usually do not need this routine.</p> <p>void svcerr_progvers(const SVCXPRT * xprt , const rpcvers_t low_vers , const rpcvers_t high_vers);</p> <p>Called when the desired version of a program is not registered with the RPC package. <i>low_vers</i> is the lowest version number, and <i>high_vers</i> is the highest version number. Service implementors usually do not need this routine.</p> <p>void svcerr_systemerr(const SVCXPRT * xprt);</p> |

Called by a service dispatch routine when it detects a system error not covered by any particular protocol. For example, if a service can no longer allocate storage, it may call this routine.

void svcerr_weakauth(const SVCXPRT * xprt);

Called by a service dispatch routine that refuses to perform a remote procedure call due to insufficient (but correct) authentication parameters. The routine calls `svcerr_auth(xprt, AUTH_TOOWEAK)`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`rpc(3N)`, `rpc_svc_calls(3N)`, `rpc_svc_create(3N)`,
`rpc_svc_reg(3N)`, `attributes(5)`

| | |
|--------------------|--|
| NAME | rpc_svc_reg, rpc_reg, svc_reg, svc_unreg, svc_auth_reg, xprt_register, xprt_unregister – library routines for registering servers |
| DESCRIPTION | These routines are a part of the RPC library which allows the RPC servers to register themselves with rpcbind() (see rpcbind(1M)), and associate the given program and version number with the dispatch function. When the RPC server receives a RPC request, the library invokes the dispatch routine with the appropriate arguments. |
| Routines | <p>See rpc(3N) for the definition of the SVCXPRT data structure.</p> <pre>#include <rpc/rpc.h></pre> <p>bool_t rpc_reg(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const rpcproc_t <i>procnum</i>, char * (* <i>procname</i>)(), const xdrproc_t <i>inproc</i>, const xdrproc_t <i>outproc</i>, const char * <i>nettype</i>);</p> <p>Register program <i>prognum</i>, procedure <i>procname</i>, and version <i>versnum</i> with the RPC service package. If a request arrives for program <i>prognum</i>, version <i>versnum</i>, and procedure <i>procnum</i>, <i>procname</i> is called with a pointer to its parameter(s); <i>procname</i> should return a pointer to its <code>static</code> result(s). The <i>arg</i> parameter to <i>procname</i> is a pointer to the (decoded) procedure argument. <i>inproc</i> is the XDR function used to decode the parameters while <i>outproc</i> is the XDR function used to encode the results. Procedures are registered on all available transports of the class <i>nettype</i>. See rpc(3N). This routine returns 0 if the registration succeeded, -1 otherwise.</p> <p>int svc_reg(const SVCXPRT * <i>xprt</i>, const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const void (* <i>dispatch</i>)(), const struct netconfig * <i>netconf</i>);</p> <p>Associates <i>prognum</i> and <i>versnum</i> with the service dispatch procedure, <i>dispatch</i>. If <i>netconf</i> is <code>NULL</code>, the service is not registered with the <code>rpcbind</code> service. For example, if a service has already been registered using some other means, such as <code>inetd</code> (see inetd(1M)), it will not need to be registered again. If <i>netconf</i> is non-zero, then a mapping of the triple [<i>prognum</i>, <i>versnum</i>, <i>netconf</i> \Rightarrow <i>nc_netid</i>] to <i>xprt</i> \Rightarrow <i>xp_ltaddr</i> is established with the local <code>rpcbind</code> service.</p> <p>The svc_reg() routine returns 1 if it succeeds, and 0 otherwise.</p> <p>void svc_unreg(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>);</p> <p>Remove from the <code>rpcbind</code> service, all mappings of the triple [<i>prognum</i>, <i>versnum</i>, <i>all-transports</i>] to network address and all mappings within the RPC service package of the double [<i>prognum</i>, <i>versnum</i>] to dispatch routines.</p> |

```
int svc_auth_reg(const int cred_flavor , const enum auth_stat (*handler)());
```

Registers the service authentication routine *handler* with the dispatch mechanism so that it can be invoked to authenticate RPC requests received with authentication type *cred_flavor* . This interface allows developers to add new authentication types to their RPC applications without needing to modify the libraries. Service implementors usually do not need this routine.

Typical service application would call **svc_auth_reg()** after registering the service and prior to calling **svc_run()** . When needed to process an RPC credential of type *cred_flavor* , the *handler* procedure will be called with two parameters (*struct svc_req * rqst* , *struct rpc_msg * msg*) and is expected to return a valid *enum auth_stat* value. There is no provision to change or delete an authentication handler once registered.

The **svc_auth_reg()** routine returns 0 if the registration is successful, 1 if *cred_flavor* already has an authentication handler registered for it, and -1 otherwise.

```
void xprt_register(const SVCXPRT * xprt );
```

After RPC service transport handle *xprt* is created, it is registered with the RPC service package. This routine modifies the global variable *svc_fdset* (see **rpc_svc_calls(3N)**). Service implementors usually do not need this routine.

```
void xprt_unregister(const SVCXPRT * xprt );
```

Before an RPC service transport handle *xprt* is destroyed, it unregisters itself with the RPC service package. This routine modifies the global variable *svc_fdset* (see **rpc_svc_calls(3N)**). Service implementors usually do not need this routine.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

inetd(1M) , **rpcbind(1M)** , **rpc(3N)** , **rpc_svc_calls(3N)** , **rpc_svc_create(3N)** , **rpc_svc_err(3N)** , **rpcbind(3N)** , **select(3C)** , **attributes(5)**

| | |
|--------------------|--|
| NAME | rpc_xdr, xdr_accepted_reply, xdr_authsys_parms, xdr_callhdr, xdr_callmsg, xdr_opaque_auth, xdr_rejected_reply, xdr_replymsg – XDR library routines for remote procedure calls |
| DESCRIPTION | These routines are used for describing the RPC messages in XDR language. They should normally be used by those who do not want to use the RPC package directly. These routines return TRUE if they succeed, FALSE otherwise. |
| Routines | <p>See rpc(3N) for the definition of the XDR data structure.</p> <pre>#include <rpc/rpc.h></pre> <p>bool_t xdr_accepted_reply(XDR * xdrs , const struct accepted_reply * ar);</p> <p>Used to translate between RPC reply messages and their external representation. It includes the status of the RPC call in the XDR language format. In the case of success, it also includes the call results.</p> <p>bool_t xdr_authsys_parms(XDR * xdrs , struct authsys_parms * aupp);</p> <p>Used for describing UNIX operating system credentials. It includes machine-name, uid, gid list, etc.</p> <p>void xdr_callhdr(XDR * xdrs , struct rpc_msg * chdr);</p> <p>Used for describing RPC call header messages. It encodes the static part of the call message header in the XDR language format. It includes information such as transaction ID, RPC version number, program and version number.</p> <p>bool_t xdr_callmsg(XDR * xdrs , struct rpc_msg * cmsg);</p> <p>Used for describing RPC call messages. This includes all the RPC call information such as transaction ID, RPC version number, program number, version number, authentication information, etc. This is normally used by servers to determine information about the client RPC call.</p> <p>bool_t xdr_opaque_auth(XDR * xdrs , struct opaque_auth * ap);</p> <p>Used for describing RPC opaque authentication information messages.</p> <p>bool_t xdr_rejected_reply(XDR * xdrs , const struct rejected_reply * rr);</p> <p>Used for describing RPC reply messages. It encodes the rejected RPC message in the XDR language format. The message could be rejected either because of version number mis-match or because of authentication errors.</p> |

bool_t xdr_replymsg(XDR * xdrs , const struct rpc_msg * rmsg);

Used for describing RPC reply messages. It translates between the RPC reply message and its external representation. This reply could be either an acceptance, rejection or NULL .

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

rpc(3N) , **xdr(3N)** , **attributes(5)**

| NAME | rstat, havedisk – get performance data from remote kernel | | | | |
|--------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lrpcsvc [library ...] #include <rpc/rpc.h> #include <rpcsvc/rstat.h> enum clnt_stat rstat(char * host, struct statstime * statp); int havedisk(char * host);</pre> | | | | |
| PROTOCOL | /usr/include/rpcsvc/rstat.x | | | | |
| DESCRIPTION | <p>These routines require that the <code>rpc.rstatd(1M)</code> daemon be configured and available on the remote system indicated by <code>host</code>. The <code>rstat()</code> protocol is used to gather statistics from remote kernel. Statistics will be available on items such as paging, swapping, and cpu utilization.</p> <p><code>rstat()</code> fills in the <code>statstime</code> structure <code>statp</code> for <code>host</code>. <code>statp</code> must point to an allocated <code>statstime</code> structure. <code>rstat()</code> returns <code>RPC_SUCCESS</code> if it was successful; otherwise a <code>enum clnt_stat</code> is returned which can be displayed using <code>clnt_perrno(3N)</code>.</p> <p><code>havedisk()</code> returns 1 if <code>host</code> has disk, 0 if it does not, and -1 if this cannot be determined.</p> <p>The following XDR routines are available in <code>librpcsvc</code>:</p> <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <pre>xdr_statstime xdr_statsvar</pre> </div> | | | | |
| ATTRIBUTES | <p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |

SEE ALSO | `rup(1)` , `rpc.rstatd(1M)` , `rpc_clnt_calls(3N)` , `attributes(5)`

| | |
|--------------------|---|
| NAME | rusers, rnusers – return information about users on remote machines |
| SYNOPSIS | <pre>cc [flag ...] file ... -lrpcsvc [library ...] #include <rpc/rpc.h> #include <rpcsvc/rusers.h> enum clnt_stat rusers(char * host, struct utmpidlearr * up); int rnusers(char * host);</pre> |
| PROTOCOL | /usr/include/rpcsvc/rusers.x |
| DESCRIPTION | <p>These routines require that the <code>rpc.rusersd(1M)</code> daemon be configured and available on the remote system indicated by <code>host</code>. The <code>rusers()</code> protocol is used to retrieve information about users logged in on the remote system.</p> <p><code>rusers()</code> fills the <code>utmpidlearr</code> structure with data about <code>host</code>, and returns 0 if successful. <code>up</code> must point to an allocated <code>utmpidlearr</code> structure. If <code>rusers()</code> returns successful it will have allocated data structures within the <code>up</code> structure, which should be freed with <code>xdr_free(3N)</code> when you no longer need them:</p> <pre>xdr_free(xdr_utmpidlearr, up);</pre> <p>On error, the returned value can be interpreted as an <code>enum clnt_stat</code> and can be displayed with <code>clnt_perror(3N)</code> or <code>clnt_sperrno(3N)</code>.</p> <p>See the header <code><rpcsvc/rusers.h></code> for a definition of <code>struct utmpidlearr</code>.</p> <p><code>rnusers()</code> returns the number of users logged on to <code>host</code> (-1 if it cannot determine that number).</p> <p>The following XDR routines are available in <code>librpcsvc</code>:</p> <pre>xdr_utmpidlearr</pre> |
| ATTRIBUTES | See <code>attributes (5)</code> for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`rusers(1)`, `rpc.rusersd(1M)`, `rpc_clnt_calls(3N)`, `xdr_free(3N)`,
`attributes(5)`

NAME rwall - write to specified remote machines

SYNOPSIS `cc [flag ...] file ... -lrpcsvc [library ...]`
`#include <rpc/rpc.h>`
`#include <rpcsvc/rwall.h>`

`enum clnt_stat rwall(char *host, char *msg);`

PROTOCOL /usr/include/rpcsvc/rwall.x

DESCRIPTION These routines require that the `rpc.rwallld(1M)` daemon be configured and available on the remote system indicated by *host*.

`rwall()` executes `wall(1M)` on *host*. The `rpc.rwallld` process on *host* prints *msg* to all users logged on to that system. `rwall()` returns `RPC_SUCCESS` if it was successful; otherwise a `enum clnt_stat` is returned which can be displayed using `clnt_perrno(3N)`.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO `rpc.rwallld(1M)`, `wall(1M)`, `rpc_clnt_calls(3N)`, `attributes(5)`

NAME | rwlock, rwlock_init, rwlock_destroy, rw_rdlock, rw_wrlock, rw_tryrdlock, rw_trywrlock, rw_unlock – multiple readers, single writer locks

SYNOPSIS

```
cc - mt [
  flag
  ... ]
file
...[
  library
  ... ]

#include <synch.h>

int rwlock_init(rwlock_t * rwp, int type, void * arg);

int rwlock_destroy(rwlock_t * rwp);

int rw_rdlock(rwlock_t * rwp);

int rw_wrlock(rwlock_t * rwp);

int rw_unlock(rwlock_t * rwp);

int rw_tryrdlock(rwlock_t * rwp);

int rw_trywrlock(rwlock_t * rwp);
```

DESCRIPTION

Many threads can have simultaneous read-only access to data, while only one thread can have write access at any given time. Multiple read access with single write access is controlled by locks, which are generally used to protect data that is frequently searched.

Readers/writer locks can synchronize threads in this process and other processes if they are allocated in writable memory and shared among cooperating processes (see `mmap(2)`), and are initialized for this purpose.

Additionally, readers/writer locks must be initialized prior to use.

rwlock_init() The readers/writer lock pointed to by *rwp* is initialized by **rwlock_init()**. A readers/writer lock is capable of having several types of behavior, which is specified by *type*. *arg* is currently not used, although a future type may define new behavior parameters by way of *arg*.

type may be one of the following:

| | |
|---------------|---|
| USYNC_PROCESS | The readers/writer lock can synchronize threads in this process and other processes. The readers/writer lock should be initialized by only one process. <i>arg</i> is ignored. A readers/writer lock initialized with this type, must be allocated in memory shared between processes, i.e. either in |
|---------------|---|

Sys V shared memory (see `shmop(2)`) or in memory mapped to a file (see `mmap(2)`). It is illegal to initialize the object this way and to not allocate it in such shared memory.

`USYNC_THREAD` The readers/writer lock can synchronize threads in this process, only. *arg* is ignored.

Additionally, readers/writer locks can be initialized by allocation in zeroed memory. A type of `USYNC_THREAD` is assumed in this case. Multiple threads must not simultaneously initialize the same readers/writer lock. And a readers/writer lock must not be re-initialized while in use by other threads.

The following are default readers/writer lock initialization (intra-process):

```
    rwlock_t rwlp;
    rwlock_init(&rwlp,
NULL
,
NULL
);
```

OR

```
    rwlock_init(&rwlp,
USYNC_THREAD
,
NULL
);
```

OR

```
    rwlock_t rwlp = DEFAULTRWLOCK;
```

The following is a customized readers/writer lock initialization (inter-process):

```
    rwlock_init(&rwlp, USYNC_PROCESS, NULL);
```

Any state associated with the readers/writer lock pointed to by *rwlp* are destroyed by `rwlock_destroy()` and the readers/writer lock storage space is not released.

`rw_rdlock()` gets a read lock on the readers/writer lock pointed to by *rwlp*. If the readers/writer lock is currently locked for writing, the calling thread blocks until the write lock is freed. Multiple threads may simultaneously hold a read lock on a readers/writer lock.

`rw_tryrdlock()` tries to get a read lock on the readers/writer lock pointed to by *rwlp*. If the readers/writer lock is locked for writing, it returns an error; otherwise, the read lock is acquired.

rw_wrllock() gets a write lock on the readers/writer lock pointed to by *rwlp* . If the readers/writer lock is currently locked for reading or writing, the calling thread blocks until all the read and write locks are freed. At any given time, only one thread may have a write lock on a readers/writer lock.

rw_trywrllock() tries to get a write lock on the readers/writer lock pointed to by *rwlp* . If the readers/writer lock is currently locked for reading or writing, it returns an error.

rw_unlock() unlocks a readers/writer lock pointed to by *rwlp* , if the readers/writer lock is locked and the calling thread holds the lock for either reading or writing. One of the other threads that is waiting for the readers/writer lock to be freed will be unblocked, provided there is other waiting threads. If the calling thread does not hold the lock for either reading or writing, no error status is returned, and the program's behavior is unknown.

RETURN VALUES

If successful, these functions return 0 . Otherwise, a non-zero value is returned to indicate the error.

ERRORS

The **rwlock_init()** function will fail if:

EINVAL *type* is invalid.

The **rw_tryrdlock()** or **rw_trywrllock()** functions will fail if:

EBUSY The reader or writer lock pointed to by *rwlp* was already locked.

These functions may fail if:

EFAULT *rwlp* or *arg* points to an illegal address.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

mmap(2) , **attributes(5)**

NOTES

These interfaces also available by way of:

```
#include <thread.h>
```

If multiple threads are waiting for a readers/writer lock, the acquisition order is random by default. However, some implementations may bias acquisition order to avoid depriving writers. The current implementation favors writers over readers.

NAME scalb – load exponent of a radix-independent floating-point number

SYNOPSIS #include <math.h>

```
double scalb(double x, double n);
```

DESCRIPTION The **scalb()** function computes $x * r^n$, where r is the radix of the machine's floating point arithmetic. When r is 2, **scalb()** is equivalent to **ldexp(3C)**.

RETURN VALUES Upon successful completion, the **scalb()** function returns $x * r^n$.

If the correct value would overflow, **scalb()** returns \pm HUGE_VAL (according to the sign of x) and sets `errno` to ERANGE.

If the correct value would underflow to 0.0, **scalb()** returns 0 and sets `errno` to ERANGE.

The **scalb()** function returns x when x is \pm Inf.

If x or n is NaN, then **scalb()** returns NaN.

For exceptional cases, **matherr(3M)** tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS The **scalb()** function will fail if:

ERANGE the correct value would overflow or underflow.

USAGE An application wishing to check for error situations should set `errno` to 0 before calling **scalb()**. If `errno` is non-zero on return, or the return value is NaN, an error has occurred.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **ldexp(3C)**, **matherr(3M)**, **attributes(5)**

NAME | scalbn – load exponent of a radix-independent floating-point number

SYNOPSIS | cc [*flag* ...] *file* ... -lm [*library* ...]
 | #include <math.h>
 |
 | double **scalbn**(double *x*, int *n*);

DESCRIPTION | The **scalbn()** function computes $x * r^n$, where *r* is the radix of the machine's floating point arithmetic.

RETURN VALUES | Upon successful completion, the **scalbn()** function returns $x * r^n$.
 | If the correct value would overflow, **scalbn()** returns \pm HUGE_VAL (according to the sign of *x*).
 | The **scalbn()** function returns *x* when *x* is \pm Inf.
 | If *x* is NaN, then **scalbn()** returns NaN.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **attributes(5)**

| | |
|----------------------|--|
| NAME | scandir, alphasort – scan a directory |
| SYNOPSIS | <pre> /usr/ucb/cc [flag ...] file ... #include <sys/types.h> #include <sys/dir.h> int scandir(dirname, namelist, select, dcomp); char * dirname ; struct direct *(* namelist []); int (* select)(.)(* dcomp)0; int alphasort(d1, d2); struct direct ** d1 , ** d2 ; </pre> |
| DESCRIPTION | <p>The scandir() function reads the directory <i>dirname</i> and builds an array of pointers to directory entries using malloc(3C). The second parameter is a pointer to an array of structure pointers. The third parameter is a pointer to a routine which is called with a pointer to a directory entry and should return a non zero value if the directory entry should be included in the array. If this pointer is NULL, then all the directory entries will be included. The last argument is a pointer to a routine which is passed to qsort(3C), which sorts the completed array. If this pointer is NULL, the array is not sorted.</p> <p>The alphasort() function is a routine that sorts the array alphabetically.</p> |
| RETURN VALUES | <p>The scandir() function returns the number of entries in the array and a pointer to the array through the parameter <i>namelist</i>. The scandir() function returns -1</p> |

if the directory cannot be opened for reading or if `malloc(3C)` cannot allocate enough memory to hold all the data structures.

USAGE The `scandir()` and `alphasort()` functions have transitional interfaces for 64-bit file offsets. See `1f64(5)` .

SEE ALSO `getdents(2)` , `malloc(3C)` , `qsort(3C)` , `readdir(3B)` , `readdir(3C)` , `1f64(5)`

NOTES Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

| | |
|--------------------|---|
| NAME | scanf, fscanf, sscanf – convert formatted input |
| SYNOPSIS | <pre>#include <stdio.h> int scanf(const char * format, ...); int fscanf(FILE* stream, const char * format, ...); int sscanf(const char * s, const char * format, ...);</pre> |
| DESCRIPTION | <p>The scanf() function reads from the standard input stream <code>stdin</code>.</p> <p>The fscanf() function reads from the named input <i>stream</i>.</p> <p>The sscanf() function reads from the string <i>s</i>.</p> <p>Each function reads bytes, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control string <i>format</i> described below, and a set of <i>pointer</i> arguments indicating where the converted input should be stored. The result is undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.</p> <p>Conversions can be applied to the <i>n</i>th argument after the <i>format</i> in the argument list, rather than to the next unused argument. In this case, the conversion character % (see below) is replaced by the sequence % <i>n</i> \$, where <i>n</i> is a decimal integer in the range [1, NL_ARGMAX]. This feature provides for the definition of format strings that select arguments in an order appropriate to specific languages. In format strings containing the % <i>n</i> \$ form of conversion specifications, it is unspecified whether numbered arguments in the argument list can be referenced from the format string more than once.</p> <p>The <i>format</i> can contain either form of a conversion specification, that is, % or % <i>n</i> \$, but the two forms cannot normally be mixed within a single <i>format</i> string. The only exception to this is that %% or %* can be mixed with the % <i>n</i> \$ form.</p> <p>The scanf() function in all its forms allows for detection of a language-dependent radix character in the input string. The radix character is defined in the program's locale (category LC_NUMERIC). In the POSIX locale, or in a locale where the radix character is not defined, the radix character defaults to a period (.).</p> <p>The format is a character string, beginning and ending in its initial shift state, if any, composed of zero or more directives. Each directive is composed of one of the following:</p> <ul style="list-style-type: none"> ■ one or more <i>white-space characters</i> (space, tab, newline, vertical-tab or form-feed characters); ■ an <i>ordinary character</i> (neither % nor a white-space character); or |

Conversion Specifications

- a *conversion specification* .

Each conversion specification is introduced by the character `%` or the character sequence `% n $` , after which the following appear in sequence:

- An optional assignment-suppressing character `*` .
- An optional non-zero decimal integer that specifies the maximum field width.
- An optional size modifier `h` , `l` (ell), or `L` indicating the size of the receiving object. The conversion characters `d` , `i` , and `n` must be preceded by `h` if the corresponding argument is a pointer to `short int` rather than a pointer to `int` , or by `l` (ell) if it is a pointer to `long int` . Similarly, the conversion characters `o` , `u` , and `x` must be preceded by `h` if the corresponding argument is a pointer to `unsigned short int` rather than a pointer to `unsigned int` , or by `l` (ell) if it is a pointer to `unsigned long int` . The conversion characters `e` , `f` , and `g` must be preceded by `l` (ell) if the corresponding argument is a pointer to `double` rather than a pointer to `float` , or by `L` if it is a pointer to `long double` . Finally, the conversion characters `c` , `s` , and `[]` must be preceded by `l` (ell) if the corresponding argument is a pointer to `wchar_t` rather than a pointer to a character type. If an `h` , `l` (ell), or `L` appears with any other conversion character, the behavior is undefined.
- A conversion character that specifies the type of conversion to be applied. The valid conversion characters are described below.

The `scanf()` functions execute each directive of the format in turn. If a directive fails, as detailed below, the function returns. Failures are described as input failures (due to the unavailability of input bytes) or matching failures (due to inappropriate input).

A directive composed of one or more white-space characters is executed by reading input until no more valid input can be read, or up to the first byte which is not a white-space character which remains unread.

A directive that is an ordinary character is executed as follows. The next byte is read from the input and compared with the byte that comprises the directive; if the comparison shows that they are not equivalent, the directive fails, and the differing and subsequent bytes remain unread.

A directive that is a conversion specification defines a set of matching input sequences, as described below for each conversion character. A conversion specification is executed in the following steps:

Input white-space characters (as specified by `isspace(3C)`)are skipped, unless the conversion specification includes a `[]` , `c` , `C` , or `n` conversion character.

An item is read from the input, unless the conversion specification includes an *n* conversion character. An input item is defined as the longest sequence of input bytes (up to any specified maximum field width, which may be measured in characters or bytes dependent on the conversion character) which is an initial subsequence of a matching sequence. The first byte, if any, after the input item remains unread. If the length of the input item is 0, the execution of the conversion specification fails; this condition is a matching failure, unless end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is an input failure.

Except in the case of a % conversion character, the input item (or, in the case of a % *n* conversion specification, the count of input bytes) is converted to a type appropriate to the conversion character. If the input item is not a matching sequence, the execution of the conversion specification fails; this condition is a matching failure. Unless assignment suppression was indicated by a * , the result of the conversion is placed in the object pointed to by the first argument following the *format* argument that has not already received a conversion result if the conversion specification is introduced by % , or in the *n* th argument if introduced by the character sequence % *n* \$. If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the space provided, the behavior is undefined.

Conversion Characters

The following conversion characters are valid:

- d Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of `strtol(3C)` with the value 10 for the *base* argument. In the absence of a size modifier, the corresponding argument must be a pointer to `int` .
- i Matches an optionally signed integer, whose format is the same as expected for the subject sequence of `strtol()` with 0 for the *base* argument. In the absence of a size modifier, the corresponding argument must be a pointer to `int` .
- o Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of `strtoul(3C)` with the value 8 for the *base* argument. In the absence of a size modifier, the corresponding argument must be a pointer to `unsigned int` .
- u Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of `strtoul()` with the value 10 for the *base* argument. In the absence of a size modifier, the corresponding argument must be a pointer to `unsigned int` .
- x Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of `strtoul()` with the value

16 for the *base* argument. In the absence of a size modifier, the corresponding argument must be a pointer to `unsigned int`.

`e`, `f`, `g` Matches an optionally signed floating-point number, whose format is the same as expected for the subject sequence of `strtod(3C)`. In the absence of a size modifier, the corresponding argument must be a pointer to `float`.

If the `printf(3S)` family of functions generates character string representations for infinity and NaN (a 7858 symbolic entity encoded in floating-point format) to support the ANSI/IEEE Std 754: 1985 standard, the `scanf()` family of functions will recognize them as input.

`s` Matches a sequence of bytes that are not white-space characters. The corresponding argument must be a pointer to the initial byte of an array of `char`, `signed char`, or `unsigned char` large enough to accept the sequence and a terminating null character code, which will be added automatically.

If an `l` (`ell`) qualifier is present, the input is a sequence of characters that begins in the initial shift state. Each character is converted to a wide-character as if by a call to the `mbrtowc(3C)` function, with the conversion state described by an `mbstate_t` object initialized to zero before the first character is converted. The corresponding argument must be a pointer to an array of `wchar_t` large enough to accept the sequence and the terminating null wide-character, which will be added automatically.

`[` Matches a non-empty sequence of characters from a set of expected characters (the *scanset*). The normal skip over white-space characters is suppressed in this case. The corresponding argument must be a pointer to the initial byte of an array of `char`, `signed char`, or `unsigned char` large enough to accept the sequence and a terminating null byte, which will be added automatically.

If an `l` (`ell`) qualifier is present, the input is a sequence of characters that begins in the initial shift state. Each character in the sequence is converted to a wide-character as if by a call to the `mbrtowc()` function, with the conversion state described by an `mbstate_t` object initialized to zero before the first character is converted. The corresponding argument must be a pointer to an array of `wchar_t` large enough to accept the sequence and the terminating null wide-character, which will be added automatically.

The conversion specification includes all subsequent characters in the *format* string up to and including the matching right square bracket (`]`)

). The characters between the square brackets (the *scanlist*) comprise the scanset, unless the character after the left square bracket is a circumflex (^), in which case the scanset contains all characters that do not appear in the scanlist between the circumflex and the right square bracket. If the conversion specification begins with [] or [^] , the right square bracket is included in the scanlist and the next right square bracket is the matching right square bracket that ends the conversion specification; otherwise the first right square bracket is the one that ends the conversion specification. If a -- is in the scanlist and is not the first character, nor the second where the first character is a ^ , nor the last character, it indicates a range of characters to be matched.

C Matches a sequence of characters of the number specified by the field width (1 if no field width is present in the conversion specification). The corresponding argument must be a pointer to the initial byte of an array of `char` , `signed char` , or `unsigned char` large enough to accept the sequence. No null byte is added. The normal skip over white-space characters is suppressed in this case.

If an `l` (ell) qualifier is present, the input is a sequence of characters that begins in the initial shift state. Each character in the sequence is converted to a wide-character as if by a call to the `mbrtowc()` function, with the conversion state described by an `mbstate_t` object initialized to zero before the first character is converted. The corresponding argument must be a pointer to an array of `wchar_t` large enough to accept the resulting sequence of wide-characters. No null wide-character is added.

P Matches the set of sequences that is the same as the set of sequences that is produced by the `%p` conversion of the corresponding `printf(3S)` functions. The corresponding argument must be a pointer to a pointer to `void` . If the input item is a value converted earlier during the same program execution, the pointer that results will compare equal to that value; otherwise the behavior of the `%p` conversion is undefined.

n No input is consumed. The corresponding argument must be a pointer to the integer into which is to be written the number of bytes read from the input so far by this call to the `scanf()` functions. Execution of a `%n` conversion specification does not increment the assignment count returned at the completion of execution of the function.

C Same as `lc` .

S Same as `ls` .

% Matches a single **%**; no conversion or assignment occurs. The complete conversion specification must be **%%**.

If a conversion specification is invalid, the behavior is undefined.

The conversion characters **E**, **G**, and **X** are also valid and behave the same as, respectively, **e**, **g**, and **x**.

If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before any bytes matching the current conversion specification (except for **%n**) have been read (other than leading white-space characters, where permitted), execution of the current conversion specification terminates with an input failure. Otherwise, unless execution of the current conversion specification is terminated with a matching failure, execution of the following conversion specification (if any) is terminated with an input failure.

Reaching the end of the string in **sscanf()** is equivalent to encountering end-of-file for **fscanf()**.

If conversion terminates on a conflicting input, the offending input is left unread in the input. Any trailing white space (including newline characters) is left unread unless matched by a conversion specification. The success of literal matches and suppressed assignments is only directly determinable via the **%n** conversion specification.

The **fscanf()** and **scanf()** functions may mark the **st_atime** field of the file associated with *stream* for update. The **st_atime** field will be marked for update by the first successful execution of **fgetc(3S)**, **fgets(3S)**, **fread(3S)**, **fscanf()**, **getc(3S)**, **getchar(3S)**, **gets(3S)**, or **scanf()** using *stream* that returns data not supplied by a prior call to **ungetc(3S)**.

RETURN VALUES

Upon successful completion, these functions return the number of successfully matched and assigned input items; this number can be 0 in the event of an early matching failure. If the input ends before the first matching failure or conversion, EOF is returned. If a read error occurs the error indicator for the stream is set, EOF is returned, and **errno** is set to indicate the error.

ERRORS

For the conditions under which the **scanf()** functions will fail and may fail, refer to **fgetc(3S)** or **fgetwc(3S)**.

In addition, **fscanf()** may fail if:

EILSEQ Input byte sequence does not form a valid character.

EINVAL There are insufficient arguments.

USAGE

If the application calling the **scanf()** functions has any objects of type `wint_t` or `wchar_t`, it must also include the header `<wchar.h>` to have these objects defined.

EXAMPLES

EXAMPLE 1 The call:

```
int i, n; float x; char name[50];

n = scanf("%d%f%s", &i, &x, name)
```

with the input line:

```
25 54.32E--1 Hamster
```

will assign to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* will contain the string Hamster.

EXAMPLE 2 The call:

```
int i; float x; char name[50];
(void) scanf("%2d%f*d %[0123456789]", &i, &x, name);
```

with input:

```
56789 0123 56a72
```

will assign 56 to *i*, 789.0 to *x*, skip 0123, and place the string 56\0 in *name*. The next call to **getchar(3S)** will return the character a.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |
| CSI | Enabled |

SEE ALSO

fgetc(3S), **fgets(3S)**, **fgetwc(3S)**, **fread(3S)**, **isspace(3C)**, **printf(3S)**, **setlocale(3C)**, **strtod(3C)**, **strtol(3C)**, **strtoul(3C)**, **wcrtomb(3C)**, **ungetc(3S)**, **attributes(5)**

| | |
|--------------------|--|
| NAME | schedctl_init, schedctl_lookup, schedctl_exit, schedctl_start, schedctl_stop – preemption control |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lsched [<i>library</i> ...] #include <schedctl.h> schedctl_t * schedctl_init(void); schedctl_t * schedctl_lookup(void); void schedctl_exit(void); void schedctl_start(schedctl_t * ptr); void schedctl_stop(schedctl_t * ptr);</pre> |
| DESCRIPTION | <p>These functions provide limited control over the scheduling of a <i>lightweight process</i> (LWP). They allow a running LWP to give a hint to the kernel that preemptions of that LWP should be avoided. The most likely use for these functions is to block preemption while holding a spinlock. Improper use of this facility, including attempts to block preemption for sustained periods of time, may result in reduced performance.</p> <p>schedctl_init() initializes preemption control for the calling LWP and returns a pointer used to refer to the data. If schedctl_init() is called more than once by the same LWP, the most recently returned pointer is the only valid one.</p> <p>schedctl_lookup() returns the currently allocated preemption control data associated with the calling LWP that was previously returned by schedctl_init(). This can be useful in programs where it is difficult to maintain local state for each LWP.</p> <p>schedctl_exit() removes the preemption control data associated with the calling LWP.</p> <p>schedctl_start() is a macro that gives a hint to the kernel scheduler that preemption should be avoided on the current LWP. The pointer passed to the macro must be the same as the pointer returned by the call to schedctl_init() by the current LWP. The behavior of the program when other values are passed is undefined.</p> |

schedctl_stop() is a macro that removes the hint that was set by **schedctl_start()**. As with **schedctl_start()**, the pointer passed to the macro must be the same as the pointer returned by the call to **schedctl_init()** by the current LWP.

schedctl_start() and **schedctl_stop()** are intended to be used to bracket short critical sections, such as the time spent holding a spinlock. Other uses, including the failure to call **schedctl_stop()** soon after calling **schedctl_start()**, may result in poor performance.

RETURN VALUES

schedctl_init() returns a pointer to a `schedctl_t` structure if the initialization was successful, or `NULL` otherwise. **schedctl_lookup()** returns a pointer to a `schedctl_t` structure if the data for that LWP was found, or `NULL` otherwise.

ERRORS

None returned.

SEE ALSO

pricntl(1), **exec(2)**, **fork(2)**, **pricntl(2)**, **thr_create(3T)**

NOTES

Preemption control is intended for use by LWPs belonging to the time-sharing (TS) and interactive (IA) scheduling classes. If used by LWPs in other scheduling classes, such as real-time (RT), no errors will be returned but **schedctl_start()** and **schedctl_stop()** will not have any effect.

Use of preemption control by unbound threads in multithreaded applications (see **thr_create(3T)**) is not supported and will result in undefined behavior.

The data used for preemption control is not copied in the child of a **fork(2)**. Thus, if a process containing LWPs using preemption control calls **fork**, and the child does not immediately call **exec(2)**, each LWP in the child must call **schedctl_init()** again prior to any future uses of **schedctl_start()** and **schedctl_stop()**. Failure to do so will result in undefined behavior.

| NAME | sched_getparam – get scheduling parameters | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <sched.h> int sched_getparam(pid_t pid, struct sched_param *param);</pre> | | | | |
| DESCRIPTION | <p>The sched_getparam() function returns the scheduling parameters of a process specified by <i>pid</i> in the <i>sched_param</i> structure pointed to by <i>param</i>.</p> <p>If a process specified by <i>pid</i> exists and if the calling process has permission, the scheduling parameters for the process whose process ID is equal to <i>pid</i> will be returned.</p> <p>If <i>pid</i> is 0, the scheduling parameters for the calling process will be returned. The behavior of the sched_getparam() function is unspecified if the value of <i>pid</i> is negative.</p> | | | | |
| RETURN VALUES | Upon successful completion, the sched_getparam() function returns 0. If the call to sched_getparam() is unsuccessful, the function returns -1 and sets <i>errno</i> to indicate the error. | | | | |
| ERRORS | <p>The sched_getparam() function will fail if:</p> <p>ENOSYS The sched_getparam() function is not supported by the system.</p> <p>EPERM The requesting process does not have permission to obtain the scheduling parameters of the specified process.</p> <p>ESRCH No process can be found corresponding to that specified by <i>pid</i>.</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | sched_getscheduler(3R) , sched_setparam(3R) , sched_setscheduler(3R) , attributes(5) , sched(5) | | | | |
| NOTES | Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned -1 and set <i>errno</i> to ENOSYS. | | | | |

| NAME | <p>sched_get_priority_max, sched_get_priority_min – get scheduling parameter limits</p> | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lrt [<i>library</i> ...] #include <sched.h> int sched_get_priority_max(int <i>policy</i>); int sched_get_priority_min(int <i>policy</i>);</pre> | | | | |
| DESCRIPTION | <p>The sched_get_priority_max() and sched_get_priority_min() functions return the appropriate maximum or minimum, respectfully, for the scheduling policy specified by <i>policy</i> .</p> <p>The value of <i>policy</i> is one of the scheduling policy values defined in <sched.h> .</p> | | | | |
| RETURN VALUES | <p>If successful, the sched_get_priority_max() and sched_get_priority_min() functions return the appropriate maximum or minimum values, respectively. If unsuccessful, they return -1 and set <code>errno</code> to indicate the error.</p> | | | | |
| ERRORS | <p>The sched_get_priority_max() and sched_get_priority_min() functions will fail if:</p> <p>EINVAL The value of the <i>policy</i> parameter does not represent a defined scheduling policy.</p> <p>ENOSYS The sched_get_priority_max() , sched_get_priority_min() and sched_rr_get_interval(3R) functions are not supported by the system.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |

SEE ALSO | `sched_getparam(3R)`, `sched_setparam(3R)`, `sched_getscheduler(3R)`,
`sched_rr_get_interval(3R)`, `sched_setscheduler(3R)`,
`attributes(5)`, `sched(5)`, `time(5)`

NOTES | Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned `-1` and set `errno` to `ENOSYS`.

NAME sched_getscheduler – get scheduling policy

SYNOPSIS

```
cc [ flag... ] file... -lrt [ library... ]
#include <sched.h>

int sched_getscheduler(pid_t pid);
```

DESCRIPTION The **sched_getscheduler()** function returns the scheduling policy of the process specified by *pid*. If the value of *pid* is negative, the behavior of the **sched_getscheduler()** function is unspecified.

The values that can be returned by **sched_getscheduler()** are defined in the header `<sched.h>` and described on the **sched_setscheduler(3R)** manual page.

If a process specified by *pid* exists and if the calling process has permission, the scheduling policy will be returned for the process whose process ID is equal to *pid*.

If *pid* is 0, the scheduling policy will be returned for the calling process.

RETURN VALUES Upon successful completion, the **sched_getscheduler()** function returns the scheduling policy of the specified process. If unsuccessful, the function returns -1 and sets `errno` to indicate the error.

ERRORS The **sched_getscheduler()** function will fail if:

- ENOSYS** The **sched_getscheduler()** function is not supported by the system.
- EPERM** The requesting process does not have permission to determine the scheduling policy of the specified process.
- ESRCH** No process can be found corresponding to that specified by *pid*.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **sched_getparam(3R)**, **sched_setparam(3R)**, **sched_setscheduler(3R)**, **attributes(5)**, **sched(5)**

NOTES

Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned `-1` and set `errno` to `ENOSYS`.

NAME sched_rr_get_interval – get execution time limits

SYNOPSIS

```
cc [ flag... ] file... -lrt [ library... ]
#include <sched.h>

int sched_rr_get_interval(pid_t pid, struct timespec *interval);
```

DESCRIPTION The **sched_rr_get_interval()** function updates the `timespec` structure referenced by the *interval* argument to contain the current execution time limit (that is, time quantum) for the process specified by *pid*. If *pid* is 0, the current execution time limit for the calling process will be returned.

RETURN VALUES If successful, the **sched_rr_get_interval()** function returns 0. Otherwise, it returns -1 and sets `errno` to indicate the error.

ERRORS The **sched_rr_get_interval()** function will fail if:

- ENOSYS** The **sched_get_priority_max(3R)**, **sched_get_priority_min(3R)**, and **sched_rr_get_interval()** functions are not supported by the system.
- ESRCH** No process can be found corresponding to that specified by *pid*.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **sched_getparam(3R)**, **sched_setparam(3R)**, **sched_get_priority_max(3R)**, **sched_getscheduler(3R)**, **sched_setscheduler(3R)**, **attributes(5)**, **sched(5)**

NOTES Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned -1 and set `errno` to `ENOSYS`.

| | |
|--------------------|---|
| NAME | sched_setparam – set scheduling parameters |
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <sched.h> int sched_setparam(pid_t pid, const struct sched_param *param);</pre> |
| DESCRIPTION | <p>The sched_setparam() function sets the scheduling parameters of the process specified by <i>pid</i> to the values specified by the <i>sched_param</i> structure pointed to by <i>param</i>. The value of the <i>sched_priority</i> member in the <i>sched_param</i> structure is any integer within the inclusive priority range for the current scheduling policy of the process specified by <i>pid</i>. Higher numerical values for the priority represent higher priorities. If the value of <i>pid</i> is negative, the behavior of the sched_setparam() function is unspecified.</p> <p>If a process specified by <i>pid</i> exists and if the calling process has permission, the scheduling parameters will be set for the process whose process ID is equal to <i>pid</i>. The real or effective user ID of the calling process must match the real or saved (from exec(2)) user ID of the target process unless the effective user ID of the calling process is 0. See intro(2).</p> <p>If <i>pid</i> is zero, the scheduling parameters will be set for the calling process.</p> <p>The target process, whether it is running or not running, resumes execution after all other runnable processes of equal or greater priority have been scheduled to run.</p> <p>If the priority of the process specified by the <i>pid</i> argument is set higher than that of the lowest priority running process and if the specified process is ready to run, the process specified by the <i>pid</i> argument preempts a lowest priority running process. Similarly, if the process calling sched_setparam() sets its own priority lower than that of one or more other non-empty process lists, then the process that is the head of the highest priority list also preempts the calling process. Thus, in either case, the originating process might not receive notification of the completion of the requested priority change until the higher priority process has executed.</p> <p>If the current scheduling policy for the process specified by <i>pid</i> is not SCHED_FIFO or SCHED_RR, including SCHED_OTHER, the result is equal to <code>prctl(P_PID, pid, PC_SETPARMS, &pcparam)</code>, where <i>pcparam</i> is an image of <i>param</i>.</p> <p>The effect of this function on individual threads is dependent on the scheduling contention scope of the threads:</p> <ul style="list-style-type: none">■ For threads with system scheduling contention scope, these functions have no effect on their scheduling. |

- For threads with process scheduling contention scope, the threads' scheduling parameters will not be affected. However, the scheduling of these threads with respect to threads in other processes may be dependent on the scheduling parameters of their process, which are governed using these functions.

If an implementation supports a two-level scheduling model in which library threads are multiplexed on top of several kernel scheduled entities, then the underlying kernel scheduled entities for the system contention scope threads will not be affected by these functions.

The underlying kernel scheduled entities for the process contention scope threads will have their scheduling parameters changed to the value specified in *param*. Kernel scheduled entities for use by process contention scope threads that are created after this call completes inherit their scheduling policy and associated scheduling parameters from the process.

This function is not atomic with respect to other threads in the process. Threads are allowed to continue to execute while this function call is in the process of changing the scheduling policy for the underlying kernel scheduled entities used by the process contention scope threads.

RETURN VALUES

If successful, the **sched_setparam()** function returns 0.

If the call to **sched_setparam()** is unsuccessful, the priority remains unchanged, and the function returns -1 and sets `errno` to indicate the error.

ERRORS

The **sched_setparam()** function will fail if:

- | | |
|---------------|--|
| EINVAL | One or more of the requested scheduling parameters is outside the range defined for the scheduling policy of the specified <i>pid</i> . |
| ENOSYS | The sched_setparam() function is not supported by the system. |
| EPERM | The requesting process does not have permission to set the scheduling parameters for the specified process, or does not have the appropriate privilege to invoke sched_setparam() . |
| ESRCH | No process can be found corresponding to that specified by <i>pid</i> . |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO `intro(2)`, `exec(2)`, `sched_getparam(3R)`, `sched_getscheduler(3R)`, `sched_setscheduler(3R)`, `attributes(5)`, `sched(5)`

NOTES Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned `-1` and set `errno` to `ENOSYS`.

| | |
|--------------------|--|
| NAME | sched_setscheduler – set scheduling policy and scheduling parameters |
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <sched.h> int sched_setscheduler(pid_t pid, int policy, const struct sched_param *param);</pre> |
| DESCRIPTION | <p>The sched_setscheduler() function sets the scheduling policy and scheduling parameters of the process specified by <i>pid</i> to <i>policy</i> and the parameters specified in the <i>sched_param</i> structure pointed to by <i>param</i>, respectively. The value of the <i>sched_priority</i> member in the <i>sched_param</i> structure is any integer within the inclusive priority range for the scheduling policy specified by <i>policy</i>. If the value of <i>pid</i> is negative, the behavior of the sched_setscheduler() function is unspecified.</p> <p>The possible values for the <i>policy</i> parameter are defined in the header file <i><sched.h></i>:</p> <p>SCHED_FIFO (realtime), First-In-First-Out; processes scheduled to this policy, if not pre-empted by a higher priority or interrupted by a signal, will proceed until completion.</p> <p>SCHED_RR (realtime), Round-Robin; processes scheduled to this policy, if not pre-empted by a higher priority or interrupted by a signal, will execute for a time period, returned by sched_rr_get_interval(3R) or by the system.</p> <p>SCHED_OTHER (time-sharing)</p> <p>If a process specified by <i>pid</i> exists and if the calling process has permission, the scheduling policy and scheduling parameters are set for the process whose process ID is equal to <i>pid</i>. The real or effective user ID of the calling process must match the real or saved (from exec(2)) user ID of the target process unless the effective user ID of the calling process is 0. See intro(2).</p> <p>If <i>pid</i> is 0, the scheduling policy and scheduling parameters are set for the calling process.</p> <p>To change the <i>policy</i> of any process to either of the real time policies SCHED_FIFO or SCHED_RR, the calling process must either have the SCHED_FIFO, or SCHED_RR policy or have an effective user ID of 0.</p> <p>The sched_setscheduler() function is considered successful if it succeeds in setting the scheduling policy and scheduling parameters of the process specified by <i>pid</i> to the values specified by <i>policy</i> and the structure pointed to by <i>param</i>, respectively.</p> <p>The effect of this function on individual threads is dependent on the scheduling contention scope of the threads:</p> |

- For threads with system scheduling contention scope, these functions have no effect on their scheduling.
- For threads with process scheduling contention scope, the threads' scheduling policy and associated parameters will not be affected. However, the scheduling of these threads with respect to threads in other processes may be dependent on the scheduling parameters of their process, which are governed using these functions.

The system supports a two-level scheduling model in which library threads are multiplexed on top of several kernel scheduled entities. The underlying kernel scheduled entities for the system contention scope threads will not be affected by these functions.

The underlying kernel scheduled entities for the process contention scope threads will have their scheduling policy and associated scheduling parameters changed to the values specified in *policy* and *param*, respectively. Kernel scheduled entities for use by process contention scope threads that are created after this call completes inherit their scheduling policy and associated scheduling parameters from the process.

This function is not atomic with respect to other threads in the process. Threads are allowed to continue to execute while this function call is in the process of changing the scheduling policy and associated scheduling parameters for the underlying kernel scheduled entities used by the process contention scope threads.

RETURN VALUES

Upon successful completion, the function returns the former scheduling policy of the specified process. If the **sched_setscheduler()** function fails to complete successfully, the policy and scheduling parameters remain unchanged, and the function returns `-1` and sets `errno` to indicate the error.

ERRORS

The **sched_setscheduler()** function will fail if:

| | |
|---------------|---|
| EINVAL | The value of <i>policy</i> is invalid, or one or more of the parameters contained in <i>param</i> is outside the valid range for the specified scheduling policy. |
| ENOSYS | The sched_setscheduler() function is not supported by the system. |
| EPERM | The requesting process does not have permission to set either or both of the scheduling parameters or the scheduling policy of the specified process. |
| ESRCH | No process can be found corresponding to that specified by <i>pid</i> . |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

priocntl(1), **intro(2)**, **exec(2)**, **priocntl(2)**,
sched_get_priority_max(3R), **sched_getparam(3R)**,
sched_getscheduler(3R), **sched_setparam(3R)**, **attributes(5)**,
sched(5)

NOTES

Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned -1 and set **errno** to **ENOSYS**.

NAME sched_yield – yield processor

SYNOPSIS

```
cc [ flag... ] file... -lrt [ library... ]
#include <sched.h>

int sched_yield(void);
```

DESCRIPTION The **sched_yield()** function forces the running thread to relinquish the processor until the process again becomes the head of its process list. It takes no arguments.

RETURN VALUES If successful, **sched_yield()** returns 0, otherwise, it returns -1, and sets **errno** to indicate the error condition.

ERRORS No errors are defined.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **attributes(5)**, **sched(5)**

| | |
|----------------------|---|
| NAME | scr_dump, scr_init, scr_restore, scr_set – write screen contents to/from a file |
| SYNOPSIS | <pre>#include <curses.h> int scr_dump(const char * filename); int scr_init(const char * filename); int scr_restore(const char * filename); int scr_set(const char * filename);</pre> |
| PARAMETERS | filename Is a pointer to the file in which screen contents are written. |
| DESCRIPTION | <p>These function perform input/output functions on a screen basis.</p> <p>The scr_dump() function writes the contents of the virtual screen, <i>curscr</i> , to <i>filename</i> .</p> <p>The scr_restore() function reads the contents of <i>filename</i> from <i>curscr</i> (which must have been written with scr_dump()). The next refresh operation restores the screen to the way it looks in <i>filename</i> .</p> <p>The scr_init() function reads the contents of <i>filename</i> and uses those contents to initialize the X/Open Curses data structures to what is actually on screen. The next refresh operation bases its updates on this data, unless the terminal has been written to since <i>filename</i> was saved or the terminfo capabilities <i>rmcup</i> and <i>nrrmc</i> are defined for the current terminal.</p> <p>The scr_set() function combines scr_restore() and scr_init() . It informs the program that the contents of the file <i>filename</i> are what is currently on the screen and that the program wants those contents on the screen.</p> |
| RETURN VALUES | On success, these functions return <i>OK</i> . Otherwise, they return <i>ERR</i> . |
| ERRORS | None. |
| SEE ALSO | delscreen(3XC) , doupdate(3XC) , endwin(3XC) , getwin(3XC) |

| | |
|----------------------|---|
| NAME | sclr, scroll, wscrl – scroll a window |
| SYNOPSIS | <pre>#include <curses.h> int sclr(int <i>n</i>); int scroll(WINDOW * <i>win</i>); int wscrl(WINDOW * <i>win</i>, int <i>n</i>);</pre> |
| PARAMETERS | <p><i>n</i> number and direction of lines to scroll</p> <p><i>win</i> pointer to the window in which to scroll</p> |
| DESCRIPTION | <p>The scroll() function scrolls the window <i>win</i> up one line. The current cursor position is not changed.</p> <p>The sclr() and wscrl() functions scroll the window <code>stdscr</code> or <i>win</i> up or down <i>n</i> lines, where <i>n</i> is a positive (scroll up) or negative (scroll down) integer.</p> <p>The scrollok(3XC) function must be enabled for these functions to work.</p> |
| RETURN VALUES | On success, these functions return <code>OK</code> . Otherwise, they return <code>ERR</code> . |
| ERRORS | None. |
| SEE ALSO | <code>clearok(3XC)</code> |

| | |
|--------------------|---|
| NAME | secure_rpc, authdes_getucred, authdes_seccreate, getnetname, host2netname, key_decryptsession, key_encryptsession, key_gendes, key_setsecret, key_secretkey_is_set, netname2host, netname2user, user2netname – library routines for secure remote procedure calls |
| DESCRIPTION | <p>RPC library routines allow C programs to make procedure calls on other machines across the network.</p> <p>RPC supports various authentication flavors. Among them are:</p> <p>AUTH_NONE (none) no authentication.</p> <p>AUTH_SYS Traditional UNIX-style authentication.</p> <p>AUTH_DES DES encryption-based authentication.</p> <p>AUTH_KERB Kerberos encryption-based authentication.</p> <p>The authdes_getucred() and authdes_seccreate() routines implement the AUTH_DES authentication flavor. The keyserver daemon keyserv (see keyserv(1M)) must be running for the AUTH_DES authentication system to work, and keylogin(1) must have been run. Only the AUTH_DES style of authentication is discussed here. For information about the AUTH_NONE and AUTH_SYS styles of authentication, refer to rpc_clnt_auth(3N). For information about the AUTH_KERB style of authentication, refer to kerberos_rpc(3N).</p> <p>The routines documented on this page are MT-Safe. See the pages of the other authentication styles for their MT-level.</p> |
| Routines | <p>See rpc(3N) for the definition of the AUTH data structure.</p> <pre>#include <rpc/rpc.h> #include <sys/types.h></pre> <p>int authdes_getucred(const struct authdes_cred * <i>adc</i> , uid_t * <i>uidp</i> , gid_t * <i>gidp</i> , short * <i>gidlenp</i> , gid_t * <i>gidlist</i>) ;</p> <p>authdes_getucred() is the first of the two routines which interface to the RPC secure authentication system known as AUTH_DES . The second is authdes_seccreate() , below. authdes_getucred() is used on the server side for converting an AUTH_DES credential, which is operating system independent, into an AUTH_SYS credential. This routine returns 1 if it succeeds, 0 if it fails.</p> <p>* <i>uidp</i> is set to the user's numerical ID associated with <i>adc</i> . * <i>gidp</i> is set to the numerical ID of the user's group. * <i>gidlist</i> contains the numerical IDs of</p> |

the other groups to which the user belongs. * *gidlenp* is set to the number of valid group ID entries in * *gidlist* (see **netname2user()** , below).

Warning: **authdes_getucred()** will fail if the *authdes_cred* structure was created with the netname of a host. In such a case, **netname2host()** should be used on the host netname in the *authdes_cred* structure to get the host name.

```
AUTH *authdes_seccreate(const char * name ,const uint_t window , const char * timehost ,const des_block * ckey );
```

authdes_seccreate() , the second of two AUTH_DES authentication routines, is used on the client side to return an authentication handle that will enable the use of the secure authentication system. The first parameter *name* is the network name, or *netname* , of the owner of the server process. This field usually represents a hostname derived from the utility routine **host2netname()** , but could also represent a user name using **user2netname()** , described below.

The second field is *window* on the validity of the client credential, given in seconds. If the difference in time between the client's clock and the server's clock exceeds *window* , the server will reject the client's credentials, and the clock will have to be resynchronized. A small window is more secure than a large one, but choosing too small of a window will increase the frequency of resynchronizations because of clock drift.

The third parameter, *timehost* , the host's name, is optional. If it is `NULL` , then the authentication system will assume that the local clock is always in sync with the *timehost* clock, and will not attempt resynchronizations. If a *timehost* is supplied, however, then the system will consult with the remote time service whenever resynchronization is required. This parameter is usually the name of the host on which the server is running.

The final parameter *ckey* is also optional. If it is `NULL` , then the authentication system will generate a random DES key to be used for the encryption of credentials. If *ckey* is supplied, then it will be used instead.

If **authdes_seccreate()** fails, it returns `NULL` .

```
int getnetname(char name [MAXNETNAMELEN+1] );
```

getnetname() returns the unique, operating system independent netname of the caller in the fixed-length array *name* . Returns 1 if it succeeds, and 0 if it fails.

```
int host2netname(char name [MAXNETNAMELEN+1], const char * host , const char * domain );
```

Convert from a domain-specific hostname *host* to an operating system independent netname. Returns 1 if it succeeds, and 0 if it fails. Inverse of **netname2host()**. If *domain* is `NULL`, **host2netname()** uses the default domain name of the machine. If *host* is `NULL`, it defaults to that machine itself. If *domain* is `NULL` and *host* is a NIS name like “host1.ssi.sun.com,” **host2netname()** uses the domain “ssi.sun.com” rather than the default domain name of the machine.

```
int key_decryptsession(const char * remotename , des_block *
deskey );
```

key_decryptsession() is an interface to the keyserver daemon, which is associated with RPC’s secure authentication system (`AUTH_DES` authentication). User programs rarely need to call it, or its associated routines **key_encryptsession()**, **key_gendes()**, and **key_setsecret()**.

key_decryptsession() takes a server netname *remotename* and a DES key *deskey*, and decrypts the key by using the the public key of the the server and the secret key associated with the effective UID of the calling process. It is the inverse of **key_encryptsession()**.

```
int key_encryptsession(const char * remotename , des_block *
deskey );
```

key_encryptsession() is a keyserver interface routine. It takes a server netname *remotename* and a DES key *deskey*, and encrypts it using the public key of the the server and the secret key associated with the effective UID of the calling process. It is the inverse of **key_decryptsession()**. This routine returns 0 if it succeeds, -1 if it fails.

```
int key_gendes(des_block * deskey );
```

key_gendes() is a keyserver interface routine. It is used to ask the keyserver for a secure conversation key. Choosing one at random is usually not good enough, because the common ways of choosing random numbers, such as using the current time, are very easy to guess. This routine returns 0 if it succeeds, -1 if it fails.

```
int key_setsecret(const char * key );
```

key_setsecret() is a keyserver interface routine. It is used to set the key for the effective UID of the calling process. This routine returns 0 if it succeeds, -1 if it fails.

```
int key_secretkey_is_set(void);
```

key_secretkey_is_set() is a keyserver interface routine that may be used to determine whether a key has been set for the effective UID of the calling process. If the keyserver has a key stored for the effective UID of the calling process, this routine returns 1. Otherwise it returns 0.

```
int netname2host(const char * name , char * host , const int hostlen );
```

Convert from an operating system independent netname *name* to a domain-specific hostname *host*. *hostlen* is the maximum size of *host*. Returns 1 if it succeeds, and 0 if it fails. Inverse of **host2netname()**.

```
int netname2user(const char * name , uid_t * uidp , gid_t * gidp , int * gidlenp , gid_t gidlist [NGRPS]);
```

Convert from an operating system independent netname to a domain-specific user ID. Returns 1 if it succeeds, and 0 if it fails. Inverse of **user2netname()**.

* *uidp* is set to the user's numerical ID associated with *name*. * *gidp* is set to the numerical ID of the user's group. *gidlist* contains the numerical IDs of the other groups to which the user belongs. * *gidlenp* is set to the number of valid group ID entries in *gidlist*.

```
int user2netname(char name [MAXNETNAMELEN+1], const uid_t uid , const char * domain );
```

Convert from a domain-specific username to an operating system independent netname. Returns 1 if it succeeds, and 0 if it fails. Inverse of **netname2user()**.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

chkey(1), **keylogin(1)**, **keyserv(1M)**, **newkey(1M)**, **kerberos_rpc(3N)**, **rpc(3N)**, **rpc_clnt_auth(3N)**, **attributes(5)**

| NAME | seekdir – set position of directory stream | | | | |
|----------------------|---|----------------|-----------------|----------|------|
| SYNOPSIS | <pre>#include <sys/types.h> #include <dirent.h> void seekdir(DIR *dirp, long int loc);</pre> | | | | |
| DESCRIPTION | <p>The seekdir() function sets the position of the next readdir(3C) operation on the directory stream specified by <i>dirp</i> to the position specified by <i>loc</i>. The value of <i>loc</i> should have been returned from an earlier call to telldir(3C). The new position reverts to the one associated with the directory stream when telldir() was performed.</p> <p>If the value of <i>loc</i> was not obtained from an earlier call to telldir() or if a call to rewinddir(3C) occurred between the call to telldir() and the call to seekdir(), the results of subsequent calls to readdir() are unspecified.</p> | | | | |
| RETURN VALUES | The seekdir() function returns no value. | | | | |
| ERRORS | No errors are defined. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe | | | | |
| SEE ALSO | opendir(3C) , readdir(3C) , rewinddir(3C) , telldir(3C) , attributes(5) | | | | |

| | |
|--------------------|--|
| NAME | select, FD_SET, FD_CLR, FD_ISSET, FD_ZERO – synchronous I/O multiplexing |
| SYNOPSIS | <pre>#include <sys/time.h> int select(int nfds, fd_set * readfds, fd_set * writefds, fd_set * errorfds, struct timeval * timeout); void FD_SET(int fd, fd_set * fdset); void FD_CLR(int fd, fd_set * fdset); int FD_ISSET(int fd, fd_set * fdset); void FD_ZERO(fd_set * fdset);</pre> |
| DESCRIPTION | <p>The select() function indicates which of the specified file descriptors is ready for reading, ready for writing, or has an error condition pending. If the specified condition is false for all of the specified file descriptors, select() blocks, up to the specified timeout interval, until the specified condition is true for at least one of the specified file descriptors.</p> <p>The select() function supports regular files, terminal and pseudo-terminal devices, STREAMS-based files, FIFOs and pipes. The behavior of select() on file descriptors that refer to other types of file is unspecified.</p> <p>The <i>nfds</i> argument specifies the range of file descriptors to be tested. The select() function tests file descriptors in the range of 0 to <i>nfds</i> - 1.</p> <p>If the <i>readfds</i> argument is not a null pointer, it points to an object of type <i>fd_set</i> that on input specifies the file descriptors to be checked for being ready to read, and on output indicates which file descriptors are ready to read.</p> <p>If the <i>writefds</i> argument is not a null pointer, it points to an object of type <i>fd_set</i> that on input specifies the file descriptors to be checked for being ready to write, and on output indicates which file descriptors are ready to write.</p> <p>If the <i>errorfds</i> argument is not a null pointer, it points to an object of type <i>fd_set</i> that on input specifies the file descriptors to be checked for error conditions pending, and on output indicates which file descriptors have error conditions pending.</p> <p>On successful completion, the objects pointed to by the <i>readfds</i>, <i>writefds</i>, and <i>errorfds</i> arguments are modified to indicate which file descriptors are ready for reading, ready for writing, or have an error condition pending, respectively. For each file descriptor less than <i>nfds</i>, the corresponding bit will be set on successful completion if it was set on input and the associated condition is true for that file descriptor.</p> |

If the *timeout* argument is not a null pointer, it points to an object of type `struct timeval` that specifies a maximum interval to wait for the selection to complete. If the *timeout* argument points to an object of type `struct timeval` whose members are 0, **select()** does not block. If the *timeout* argument is a null pointer, **select()** blocks until an event causes one of the masks to be returned with a valid (non-zero) value. If the time limit expires before any event occurs that would cause one of the masks to be set to a non-zero value, **select()** completes successfully and returns 0.

If the *readfs*, *writefs*, and *errorfds* arguments are all null pointers and the *timeout* argument is not a null pointer, **select()** blocks for the time specified, or until interrupted by a signal. If the *readfs*, *writefs*, and *errorfds* arguments are all null pointers and the *timeout* argument is a null pointer, **select()** blocks until interrupted by a signal.

File descriptors associated with regular files always select true for ready to read, ready to write, and error conditions.

On failure, the objects pointed to by the *readfs*, *writefs*, and *errorfds* arguments are not modified. If the timeout interval expires without the specified condition being true for any of the specified file descriptors, the objects pointed to by the *readfs*, *writefs*, and *errorfds* arguments have all bits set to 0.

A file descriptor for a socket that is listening for connections will indicate that it is ready for reading, when connections are available. A file descriptor for a socket that is connecting asynchronously will indicate that it is ready for writing, when a connection has been established.

Selecting true for reading on a socket descriptor upon which a **listen(3N)** call has been performed indicates that a subsequent **accept(3N)** call on that descriptor will not block.

File descriptor masks of type `fd_set` can be initialized and tested with the macros **FD_CLR()**, **FD_ISSET()**, **FD_SET()**, and **FD_ZERO()**.

FD_CLR (*fd* , & *fdset*) Clears the bit for the file descriptor *fd* in the file descriptor set *fdset*.

FD_ISSET (*fd* , & *fdset*) Returns a non-zero value if the bit for the file descriptor *fd* is set in the file descriptor set pointed to by *fdset*, and 0 otherwise.

FD_SET (*fd* , & *fdset*) Sets the bit for the file descriptor *fd* in the file descriptor set *fdset*.

FD_ZERO (& *fdset*) Initializes the file descriptor set *fdset* to have zero bits for all file descriptors.

The behavior of these macros is undefined if the *fd* argument is less than 0 or greater than or equal to `FD_SETSIZE`.

RETURN VALUES

The **FD_CLR()** , **FD_SET()** , and **FD_ZERO()** macros return no value. The **FD_ISSET()** macro returns a non-zero value if the bit for the file descriptor *fd* is set in the file descriptor set pointed to by *fdset* , and 0 otherwise.

On successful completion, **select()** returns the total number of bits set in the bit masks. Otherwise, -1 is returned, and `errno` is set to indicate the error.

ERRORS

The **select()** function will fail if:

- EBADF** One or more of the file descriptor sets specified a file descriptor that is not a valid open file descriptor.
- EINTR** The **select()** function was interrupted before any of the selected events occurred and before the timeout interval expired.
- If `SA_RESTART` has been set for the interrupting signal, it is implementation-dependent whether **select()** restarts or returns with `EINTR` .
- EINVAL** An invalid timeout interval was specified.
- EINVAL** The *nfds* argument is less than 0, or greater than or equal to `FD_SETSIZE` .
- EINVAL** One of the specified file descriptors refers to a `STREAM` or multiplexer that is linked (directly or indirectly) downstream from a multiplexer.
- EINVAL** A component of the pointed-to time limit is outside the acceptable range: `t_sec` must be between 0 and 10^8 , inclusive. `t_usec` must be greater than or equal to 0 , and less than 10^6 .

USAGE

The `poll(2)` function is preferred over this function. It must be used when the number of file descriptors exceeds `FD_SETSIZE` .

The use of a timeout does not affect any pending timers set up by `alarm(2)` , `ualarm(3C)` or `setitimer(2)` .

On successful completion, the object pointed to by the *timeout* argument may be modified.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

alarm(2), **fcntl(2)**, **poll(2)**, **read(2)**, **setitimer(2)**, **write(2)**,
accept(3N), **listen(3N)**, **ualarm(3C)**, **attributes(5)**

NOTES

The default value for `FD_SETSIZE` (currently 1024) is larger than the default limit on the number of open files. To accommodate 32-bit applications that wish to use a larger number of open files with **select()**, it is possible to increase this size at compile time by providing a larger definition of `FD_SETSIZE` before the inclusion of `<sys/types.h>`. The maximum supported size for `FD_SETSIZE` is 65536. The default value is already 65536 for 64-bit applications.

| | | | |
|--------------------|--|---------------|---|
| NAME | semaphore, sema_init, sema_destroy, sema_wait, sema_trywait, sema_post – semaphores | | |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ...- lthread - lc [<i>library</i> ...] #include <synch.h> int sema_init(sema_t * <i>sp</i>, unsigned int <i>count</i>, int <i>type</i>, void * <i>arg</i>); int sema_destroy(sema_t * <i>sp</i>); int sema_wait(sema_t * <i>sp</i>); int sema_trywait(sema_t * <i>sp</i>); int sema_post(sema_t * <i>sp</i>);</pre> | | |
| DESCRIPTION | <p>A semaphore is a non-negative integer count and is generally used to coordinate access to resources. The initial semaphore count is set to the number of free resources, then threads slowly increment and decrement the count as resources are added and removed. If the semaphore count drops to zero, which means no available resources, threads attempting to decrement the semaphore will block until the count is greater than zero.</p> <p>Semaphores can synchronize threads in this process and other processes if they are allocated in writable memory and shared among the cooperating processes (see <code>mmap(2)</code>), and have been initialized for this purpose.</p> <p>Semaphores must be initialized before use; semaphores pointed to by <i>sp</i> to <i>count</i> are initialized by <code>sema_init()</code>. <i>type</i> can assign several different types of behavior to a semaphore. No current type uses <i>arg</i> although it may be used in the future.</p> <p><i>type</i> may be one of the following:</p> <table border="0"> <tr> <td style="padding-right: 20px;">USYNC_PROCESS</td> <td>The semaphore can synchronize threads in this process and other processes. Initializing the semaphore should be done by only one process. A semaphore initialized with this type must be allocated in memory shared between processes, i.e. either in Sys V shared memory (see <code>shmop(2)</code>), or in memory mapped to a file (see <code>mmap(2)</code>). It is illegal to initialize the object this way and to</td> </tr> </table> | USYNC_PROCESS | The semaphore can synchronize threads in this process and other processes. Initializing the semaphore should be done by only one process. A semaphore initialized with this type must be allocated in memory shared between processes, i.e. either in Sys V shared memory (see <code>shmop(2)</code>), or in memory mapped to a file (see <code>mmap(2)</code>). It is illegal to initialize the object this way and to |
| USYNC_PROCESS | The semaphore can synchronize threads in this process and other processes. Initializing the semaphore should be done by only one process. A semaphore initialized with this type must be allocated in memory shared between processes, i.e. either in Sys V shared memory (see <code>shmop(2)</code>), or in memory mapped to a file (see <code>mmap(2)</code>). It is illegal to initialize the object this way and to | | |

not allocate it in such shared memory. *arg* is ignored.

USYNC_THREAD

The semaphore can synchronize threads only in this process. *arg* is ignored.

A semaphore must not be simultaneously initialized by multiple threads, nor re-initialized while in use by other threads.

Default semaphore initialization (intra-process):

```
sema_t sp;
int count = 1;
sema_init(&sp, count, NULL, NULL);
```

\011
or

```
\011sema_init(&sp, count, USYNC_THREAD, NULL);
```

Customized semaphore initialization (inter-process):

```
\011sema_t sp; \011int count = 1; \011sema_init(&sp, count, USYNC_PROCESS, NULL);
```

sema_destroy() destroys any state related to the semaphore pointed to by *sp*. The semaphore storage space is not released.

sema_wait() blocks the calling thread until the semaphore count pointed to by *sp* is greater than zero, and then it atomically decrements the count.

sema_trywait() atomically decrements the semaphore count pointed to by *sp*, if the count is greater than zero; otherwise, it returns an error.

sema_post() atomically increments the semaphore count pointed to by *sp*. If there are any threads blocked on the semaphore, one will be unblocked.

The semaphore functionality described on this man page is for the Solaris threads implementation. For the POSIX-compliant semaphore interface documentation, see [sem_open\(3R\)](#), [sem_init\(3R\)](#), [sem_wait\(3R\)](#), [sem_post\(3R\)](#), [sem_getvalue\(3R\)](#), [sem_unlink\(3R\)](#), [sem_close\(3R\)](#), [sem_destroy\(3R\)](#)).

RETURN VALUES

Upon successful completion, 0 is returned; otherwise, a non-zero value indicates an error.

ERRORS

These functions fail and return the corresponding value if any of the following conditions are detected:

EINVAL Invalid argument.

EFAULT *sp* or *arg* points to an illegal address.
The **sema_wait()** function fails and returns the corresponding value if any of the following conditions are detected:

EINTR The wait was interrupted by a signal or **fork()** .
The **sema_trywait()** function fails and returns the corresponding value if any of the following conditions are detected:

EBUSY The semaphore pointed to by *sp* has a zero count.
The **sema_post()** function fails and returns the corresponding value if any of the following conditions are detected:

EOVERFLOW The semaphore value pointed to by *sp* exceeds SEM_VALUE_MAX.

EXAMPLES

EXAMPLE 1 The customer waiting-line in a bank is analogous to the synchronization scheme of a semaphore using **sema_wait()** and **sema_trywait()** :

```

/* cc [ flag ... ] file ... --lthread [ library ... ] */
#include <errno.h>
#define TELLERS 10
sema_t    tellers;    /* semaphore */
int banking_hours(), deposit_withdrawal;
void*customer(), do_business(), skip_banking_today();
...

sema_init(&tellers, TELLERS, USYNC_THREAD, NULL);
/* 10 tellers available */
while(banking_hours())
    pthread_create(NULL, NULL, customer, deposit_withdrawal);
...

void *
customer(int deposit_withdrawal)
{
    int this_customer, in_a_hurry = 50;
    this_customer = rand() % 100;

    if (this_customer == in_a_hurry) {
        if (sema_trywait(&tellers) != 0)
            if (errno == EAGAIN){ /* no teller available */
                skip_banking_today(this_customer);
                return;
            } /* else go immediately to available teller and
                decrement tellers */
    }
    else
        sema_wait(&tellers); /* wait for next teller, then proceed,
                               and decrement tellers */

    do_business(deposit_withdrawal);
    sema_post(&tellers); /* increment tellers;
                           this_customer's teller

```

```
is now available */
```

```
}
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------|
| MT-Level | Async-Signal-Safe |

SEE ALSO

mmap(2), **shmop(2)**, **sem_close(3R)**, **sem_destroy(3R)**,
sem_getvalue(3R), **sem_init(3R)**, **sem_open(3R)**, **sem_post(3R)**,
sem_unlink(3R), **sem_wait(3R)**, **attributes(5)**, **standards(5)**

NOTES

These interfaces are also available by way of:

```
#include <thread.h>
```

If multiple threads are waiting for a semaphore, by default, there is no defined order of unblocking.

USYNC_THREAD does not support multiple mappings to the same logical synch object. If you need to **mmap()** a synch object to different locations within the same address space, then the synch object should be initialized as a shared object **USYNC_PROCESS** for Solaris, and **PTHREAD_PROCESS_PRIVATE** for POSIX.

| NAME | sem_close – close a named semaphore | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <semaphore.h> int sem_close(sem_t *sem);</pre> | | | | |
| DESCRIPTION | <p>The sem_close() function is used to indicate that the calling process is finished using the named semaphore indicated by <i>sem</i>. The effects of calling sem_close() for an unnamed semaphore (one created by sem_init(3R)) are undefined. The sem_close() function deallocates (that is, make available for reuse by a subsequent sem_open(3R) by this process) any system resources allocated by the system for use by this process for this semaphore. The effect of subsequent use of the semaphore indicated by <i>sem</i> by this process is undefined. If the semaphore has not been removed with a successful call to sem_unlink(3R), then sem_close() has no effect on the state of the semaphore. If the sem_unlink(3R) function has been successfully invoked for <i>name</i> after the most recent call to sem_open(3R) with O_CREAT for this semaphore, then when all processes that have opened the semaphore close it, the semaphore is no longer be accessible.</p> | | | | |
| RETURN VALUES | If successful, sem_close() returns 0, otherwise it returns -1 and sets <i>errno</i> to indicate the error. | | | | |
| ERRORS | <p>The sem_close() function will fail if:</p> <p>EINVAL The <i>sem</i> argument is not a valid semaphore descriptor.</p> <p>ENOSYS The sem_close() function is not supported by the system.</p> | | | | |
| USAGE | The sem_close() function should not be called for an unnamed semaphore initialized by sem_init(3R) . | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | sem_init(3R) , sem_open(3R) , sem_unlink(3R) , attributes(5) | | | | |
| NOTES | Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned -1 and set <i>errno</i> to ENOSYS . | | | | |

| NAME | sem_destroy – destroy an unnamed semaphore | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <semaphore.h> int sem_destroy(sem_t *sem);</pre> | | | | |
| DESCRIPTION | <p>The sem_destroy() function is used to destroy the unnamed semaphore indicated by <i>sem</i>. Only a semaphore that was created using sem_init(3R) may be destroyed using sem_destroy(); the effect of calling sem_destroy() with a named semaphore is undefined. The effect of subsequent use of the semaphore <i>sem</i> is undefined until <i>sem</i> is re-initialized by another call to sem_init(3R).</p> <p>It is safe to destroy an initialised semaphore upon which no threads are currently blocked. The effect of destroying a semaphore upon which other threads are currently blocked is undefined.</p> | | | | |
| RETURN VALUES | If successful, sem_destroy() returns 0, otherwise it returns -1 and sets <i>errno</i> to indicate the error. | | | | |
| ERRORS | <p>The sem_destroy() function will fail if:</p> <p>EINVAL The <i>sem</i> argument is not a valid semaphore.</p> <p>ENOSYS The sem_destroy() function is not supported by the system.</p> <p>The sem_destroy() function may fail if:</p> <p>EBUSY There are currently processes (or LWPs or threads) blocked on the semaphore.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | sem_init(3R) , sem_open(3R) , attributes(5) | | | | |

NAME sem_getvalue – get the value of a semaphore

SYNOPSIS

```
cc [ flag... ] file... -lrt [ library... ]
#include <semaphore.h>

int sem_getvalue(sem_t *sem, int *sval);
```

DESCRIPTION

The **sem_getvalue()** function updates the location referenced by the *sval* argument to have the value of the semaphore referenced by *sem* without affecting the state of the semaphore. The updated value represents an actual semaphore value that occurred at some unspecified time during the call, but it need not be the actual value of the semaphore when it is returned to the calling process.

If *sem* is locked, then the value returned by **sem_getvalue()** is either zero or a negative number whose absolute value represents the number of processes waiting for the semaphore at some unspecified time during the call.

The value set in *sval* may be 0 or positive. If *sval* is 0, there may be other processes (or LWPs or threads) waiting for the semaphore; if *sval* is positive, no processed is waiting.

RETURN VALUES

Upon successful completion, **sem_getvalue()** returns 0. Otherwise, it returns -1 and sets *errno* to indicate the error.

ERRORS

The **sem_getvalue()** function will fail if:

EINVAL The *sem* argument does not refer to a valid semaphore.

ENOSYS The **sem_getvalue()** function is not supported by the system.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **sem_post(3R)**, **sem_wait(3R)**, **attributes(5)**

| | |
|----------------------|--|
| NAME | sem_init – initialize an unnamed semaphore |
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <semaphore.h> int sem_init(sem_t *sem, int pshared, unsigned int value);</pre> |
| DESCRIPTION | <p>The sem_init() function is used to initialize the unnamed semaphore referred to by <i>sem</i>. The value of the initialized semaphore is <i>value</i>. Following a successful call to sem_init(), the semaphore may be used in subsequent calls to sem_wait(3R), sem_trywait(3R), sem_post(3R), and sem_destroy(3R). This semaphore remains usable until the semaphore is destroyed.</p> <p>If the <i>pshared</i> argument has a non-zero value, then the semaphore is shared between processes; in this case, any process that can access the semaphore <i>sem</i> can use <i>sem</i> for performing sem_wait(3R), sem_trywait(3R), sem_post(3R), and sem_destroy(3R) operations.</p> <p>Only <i>sem</i> itself may be used for performing synchronization. The result of referring to copies of <i>sem</i> in calls to sem_wait(3R), sem_trywait(3R), sem_post(3R), and sem_destroy(3R), is undefined.</p> <p>If the <i>pshared</i> argument is zero, then the semaphore is shared between threads of the process; any thread in this process can use <i>sem</i> for performing sem_wait(3R), sem_trywait(3R), sem_post(3R), and sem_destroy(3R) operations. The use of the semaphore by threads other than those created in the same process is undefined.</p> <p>Attempting to initialize an already initialized semaphore results in undefined behavior.</p> |
| RETURN VALUES | Upon successful completion, the function initializes the semaphore in <i>sem</i> . Otherwise, it returns -1 and sets <code>errno</code> to indicate the error. |
| ERRORS | <p>The sem_init() function will fail if:</p> <p>EINVAL The <i>value</i> argument exceeds <code>SEM_VALUE_MAX</code>.</p> <p>ENOSPC A resource required to initialize the semaphore has been exhausted, or the resources have reached the limit on semaphores (<code>SEM_NSEMS_MAX</code>).</p> <p>ENOSYS The sem_init() function is not supported by the system.</p> <p>EPERM The process lacks the appropriate privileges to initialize the semaphore.</p> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`sem_destroy(3R)`, `sem_post(3R)`, `sem_wait(3R)`, `attributes(5)`

| | |
|--------------------|---|
| NAME | sem_open - initialize/open a named semaphore |
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <semaphore.h> sem_t *sem_open(const char *name, int oflag, /* unsigned long mode, unsigned int value */ ...);</pre> |
| DESCRIPTION | <p>The sem_open() function establishes a connection between a named semaphore and a process (or LWP or thread). Following a call to sem_open() with semaphore name <i>name</i>, the process may reference the semaphore associated with <i>name</i> using the address returned from the call. This semaphore may be used in subsequent calls to sem_wait(3R), sem_trywait(3R), sem_post(3R), and sem_close(3R). The semaphore remains usable by this process until the semaphore is closed by a successful call to sem_close(3R), _exit(2), or one of the exec functions.</p> <p>The <i>oflag</i> argument controls whether the semaphore is created or merely accessed by the call to sem_open(). The following flag bits may be set in <i>oflag</i>:</p> <p>O_CREAT This flag is used to create a semaphore if it does not already exist. If O_CREAT is set and the semaphore already exists, then O_CREAT has no effect, except as noted under O_EXCL. Otherwise, sem_open() creates a named semaphore. The O_CREAT flag requires a third and a fourth argument: <i>mode</i>, which is of type <code>mode_t</code>, and <i>value</i>, which is of type <code>unsigned int</code>. The semaphore is created with an initial value of <i>value</i>. Valid initial values for semaphores are less than or equal to <code>SEM_VALUE_MAX</code>.</p> <p>The user ID of the semaphore is set to the effective user ID of the process; the group ID of the semaphore is set to a system default group ID or to the effective group ID of the process. The permission bits of the semaphore are set to the value of the <i>mode</i> argument except those set in the file mode creation mask of the process (see umask(2)). When bits in <i>mode</i> other than the file permission bits are specified, the effect is unspecified.</p> <p>After the semaphore named <i>name</i> has been created by sem_open() with the O_CREAT flag, other processes can connect to the semaphore by calling sem_open() with the same value of <i>name</i>.</p> <p>O_EXCL If O_EXCL and O_CREAT are set, sem_open() fails if the semaphore <i>name</i> exists. The check for the existence of the semaphore and the creation of the semaphore if it does not exist are atomic with respect to other processes executing</p> |

sem_open() with `O_EXCL` and `O_CREAT` set. If `O_EXCL` is set and `O_CREAT` is not set, the effect is undefined.

If flags other than `O_CREAT` and `O_EXCL` are specified in the *oflag* parameter, the effect is unspecified.

The *name* argument points to a string naming a semaphore object. It is unspecified whether the name appears in the file system and is visible to functions that take pathnames as arguments. The *name* argument conforms to the construction rules for a pathname. The first character of *name* must be a slash (/) character and the remaining characters of *name* cannot include any slash characters. For maximum portability, *name* should include no more than 14 characters, but this limit is not enforced.

If a process makes multiple successful calls to **sem_open()** with the same value for *name*, the same semaphore address is returned for each such successful call, provided that there have been no calls to **sem_unlink(3R)** for this semaphore.

References to copies of the semaphore produce undefined results.

RETURN VALUES

Upon successful completion, the function returns the address of the semaphore. Otherwise, it will return a value of `SEM_FAILED` and set `errno` to indicate the error. The symbol `SEM_FAILED` is defined in the header `<semaphore.h>`. No successful return from **sem_open()** will return the value `SEM_FAILED`.

ERRORS

If any of the following conditions occur, the **sem_open()** function will return `SEM_FAILED` and set `errno` to the corresponding value:

| | |
|---------------|--|
| EACCES | The named semaphore exists and the <code>O_RDWR</code> permissions are denied, or the named semaphore does not exist and permission to create the named semaphore is denied. |
| EEXIST | <code>O_CREAT</code> and <code>O_EXCL</code> are set and the named semaphore already exists. |
| EINTR | The sem_open() function was interrupted by a signal. |
| EINVAL | The sem_open() operation is not supported for the given name, or <code>O_CREAT</code> was set in <i>oflag</i> and <i>value</i> is greater than <code>SEM_VALUE_MAX</code> . |
| EMFILE | The number of open semaphore descriptors in this process exceeds <code>SEM_NSEMS_MAX</code> , or the number of open file descriptors in this process exceeds <code>OPEN_MAX</code> . |

| | |
|---------------------|--|
| ENAMETOOLONG | The length of <i>name</i> string exceeds <code>PATH_MAX</code> , or a pathname component is longer than <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect. |
| ENFILE | Too many semaphores are currently open in the system. |
| ENOENT | <code>O_CREAT</code> is not set and the named semaphore does not exist. |
| ENOSPC | There is insufficient space for the creation of the new named semaphore. |
| ENOSYS | The <code>sem_open()</code> function is not supported by the system. |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`exec(2)`, `exit(2)`, `umask(2)`, `sem_close(3R)`, `sem_post(3R)`, `sem_unlink(3R)`, `sem_wait(3R)`, `sysconf(3C)`, `attributes(5)`

NOTES

Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned `(sem_t *)-1` and set `errno` to `ENOSYS`.

| | |
|----------------------|---|
| NAME | sem_post – increment the count of a semaphore |
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <semaphore.h> int sem_post(sem_t *sem);</pre> |
| DESCRIPTION | <p>The sem_post() function unlocks the semaphore referenced by <i>sem</i> by performing a semaphore unlock operation on that semaphore.</p> <p>If the semaphore value resulting from this operation is positive, then no threads were blocked waiting for the semaphore to become unlocked; the semaphore value is simply incremented.</p> <p>If the value of the semaphore resulting from this operation is 0, then one of the threads blocked waiting for the semaphore will be allowed to return successfully from its call to sem_wait(3R). If the symbol <code>_POSIX_PRIORITY_SCHEDULING</code> is defined, the thread to be unblocked will be chosen in a manner appropriate to the scheduling policies and parameters in effect for the blocked threads. In the case of the schedulers <code>SCHED_FIFO</code> and <code>SCHED_RR</code>, the highest priority waiting thread will be unblocked, and if there is more than one highest priority thread blocked waiting for the semaphore, then the highest priority thread that has been waiting the longest will be unblocked. If the symbol <code>_POSIX_PRIORITY_SCHEDULING</code> is not defined, the choice of a thread to unblock is unspecified.</p> |
| RETURN VALUES | If successful, sem_post() returns 0; otherwise it returns -1 and sets <code>errno</code> to indicate the error. |
| ERRORS | <p>The sem_post() function will fail if:</p> <p>EINVAL The <i>sem</i> argument does not refer to a valid semaphore.</p> <p>ENOSYS The sem_post() function is not supported by the system.</p> <p>E_OVERFLOW The semaphore value exceeds <code>SEM_VALUE_MAX</code>.</p> |
| USAGE | The sem_post() function is reentrant with respect to signals and may be invoked from a signal-catching function. The semaphore functionality described on this manual page is for the POSIX (see standards(5)) threads implementation. For the documentation of the Solaris threads interface, see semaphore(3T) . |
| EXAMPLES | EXAMPLE 1 See sem_wait(3R) . |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------|
| MT-Level | Async-Signal-Safe |

SEE ALSO `sched_setscheduler(3R)`, `sem_wait(3R)`, `semaphore(3T)`,
`attributes(5)`, `standards(5)`

NOTES Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned `-1` and set `errno` to `ENOSYS`.

NAME sem_unlink - remove a named semaphore

SYNOPSIS
 cc [*flag...*] *file...* -l`rt` [*library...*]
 #include <semaphore.h>

```
int sem_unlink(const char *name);
```

DESCRIPTION
 The `sem_unlink()` function removes the semaphore named by the string *name*. If the semaphore named by *name* is currently referenced by other processes, then `sem_unlink()` has no effect on the state of the semaphore. If one or more processes have the semaphore open when `sem_unlink()` is called, destruction of the semaphore is postponed until all references to the semaphore have been destroyed by calls to `sem_close(3R)`, `_exit(2)`, or one of the `exec` functions (see `exec(2)`). Calls to `sem_open(3R)` to re-create or re-connect to the semaphore refer to a new semaphore after `sem_unlink()` is called. The `sem_unlink()` call does not block until all references have been destroyed; it returns immediately.

RETURN VALUES
 Upon successful completion, `sem_unlink()` returns 0. Otherwise, the semaphore is not changed and the function returns a value of -1 and sets `errno` to indicate the error.

- ERRORS**
 The `sem_unlink()` function will fail if:
- EACCES** Permission is denied to unlink the named semaphore.
 - ENAMETOOLONG** The length of *name* string exceeds `PATH_MAX`, or a pathname component is longer than `NAME_MAX` while `_POSIX_NO_TRUNC` is in effect.
 - ENOENT** The named semaphore does not exist.
 - ENOSYS** The `sem_unlink()` function is not supported by the system.

ATTRIBUTES
 See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO `exec(2)`, `exit(2)`, `sem_close(3R)`, `sem_open(3R)`, `attributes(5)`

NOTES

Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned `-1` and set `errno` to `ENOSYS`.

| | |
|----------------------|---|
| NAME | sem_wait, sem_trywait – acquire or wait for a semaphore |
| SYNOPSIS | <pre>cc [flag ...] file ... -lrt [library ...] #include <semaphore.h> int sem_wait(sem_t * sem); int sem_trywait(sem_t * sem);</pre> |
| DESCRIPTION | <p>The sem_wait() function locks the semaphore referenced by <i>sem</i> by performing a semaphore lock operation on that semaphore. If the semaphore value is currently zero, then the calling thread will not return from the call to sem_wait() until it either locks the semaphore or the call is interrupted by a signal. The sem_trywait() function locks the semaphore referenced by <i>sem</i> only if the semaphore is currently not locked; that is, if the semaphore value is currently positive. Otherwise, it does not lock the semaphore.</p> <p>Upon successful return, the state of the semaphore is locked and remains locked until the sem_post(3R) function is executed and returns successfully.</p> <p>The sem_wait() function is interruptible by the delivery of a signal.</p> |
| RETURN VALUES | <p>The sem_wait() and sem_trywait() functions return 0 if the calling process successfully performed the semaphore lock operation on the semaphore designated by <i>sem</i>. If the call was unsuccessful, the state of the semaphore is unchanged, and the function returns -1 and sets <i>errno</i> to indicate the error.</p> |
| ERRORS | <p>The sem_wait() and sem_trywait() functions will fail if:</p> <p>EINVAL The <i>sem</i> function does not refer to a valid semaphore.</p> <p>ENOSYS The sem_wait() and sem_trywait() functions are not supported by the system.</p> <p>The sem_trywait() function will fail if:</p> <p>EAGAIN The semaphore was already locked, so it cannot be immediately locked by the sem_trywait() operation.</p> <p>The sem_wait() and sem_trywait() functions may fail if:</p> |

EDEADLK A deadlock condition was detected; that is, two separate processes are waiting for an available resource to be released via a semaphore "held" by the other process.

EINTR A signal interrupted this function.

USAGE

Realtime applications may encounter priority inversion when using semaphores. The problem occurs when a high priority thread "locks" (that is, waits on) a semaphore that is about to be "unlocked" (that is, posted) by a low priority thread, but the low priority thread is preempted by a medium priority thread. This scenario leads to priority inversion; a high priority thread is blocked by lower priority threads for an unlimited period of time. During system design, realtime programmers must take into account the possibility of this kind of priority inversion. They can deal with it in a number of ways, such as by having critical sections that are guarded by semaphores execute at a high priority, so that a thread cannot be preempted while executing in its critical section.

EXAMPLES

EXAMPLE 1 The customer waiting-line in a bank may be analogous to the synchronization scheme of a semaphore utilizing `sem_wait()` and `sem_trywait()` :

```
/* cc [ flag ... ]file ... --lrt --lthread [ library ... ]*/

#include <errno.h>
#define TELLERS 10
sem_t bank_line; /* semaphore */
int banking_hours(), deposit_withdrawal;
void *customer(), do_business(), skip_banking_today();
thread_t tid;
...

sem_init(&bank_line,TRUE,TELLERS); /* 10 tellers available */
while(banking_hours())
    thr_create(NULL, NULL, customer, (void *)deposit_withdrawal,
              THREAD_NEW_LWP, &tid);
...

void *
customer(deposit_withdrawal)
void *deposit_withdrawal;
{
    int this_customer, in_a_hurry = 50;
    this_customer = rand() % 100;
    if (this_customer == in_a_hurry) {
        if (sem_trywait(&bank_line) != 0)
            if (errno == EAGAIN) { /* no teller available */
                skip_banking_today(this_customer);
                return;
            } /*else go immediately to available teller
              & decrement bank_line*/
    }
}
```

```

else
    sem_wait(&bank_line); /* wait for next teller,
                          then proceed, and decrement bank_line */
do_business((int *)deposit_withdrawal);
sem_getvalue(&bank_line,&num_tellers);
sem_post(&bank_line); /* increment bank_line;
                       this_customer's teller
is now available */
}

```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

sem_post(3R), **attributes(5)**

| | | | |
|--------------------|---|---------|---|
| NAME | send, sendto, sendmsg – send a message from a socket | | |
| SYNOPSIS | <pre> cc [flag ...] file ... -lsocket -lnsl [library ...] #include <sys/types.h> #include <sys/socket.h> ssize_t send(int s, const void * msg, size_t len, int flags); ssize_t sendto(int s, const void * msg, size_t len, int flags, const struct sockaddr * to, socklen_t tolen); ssize_t sendmsg(int s, const struct msghdr * msg, int flags); </pre> | | |
| DESCRIPTION | <p>send() , sendto() , and sendmsg() are used to transmit a message to another transport end-point. send() may be used only when the socket is in a <i>connected</i> state, while sendto() and sendmsg() may be used at any time. <i>s</i> is a socket created with socket(3N) .</p> <p>The address of the target is given by <i>to</i> with <i>tolen</i> specifying its size. The length of the message is given by <i>len</i> . If the message is too long to pass atomically through the underlying protocol, then the error EMSGSIZE is returned, and the message is not transmitted.</p> <p>A return value of -1 indicates locally detected errors only. It does not implicitly mean the message was not delivered.</p> <p>If the socket does not have enough buffer space available to hold the message being sent, send() blocks, unless the socket has been placed in non-blocking I/O mode (see fcntl(2)). The select(3C) or poll(2) call may be used to determine when it is possible to send more data.</p> <p>The <i>flags</i> parameter is formed from the bitwise OR of zero or more of the following:</p> <table border="0" style="width: 100%;"> <tr> <td style="vertical-align: top; padding-right: 20px;">MSG_OOB</td> <td>Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only SOCK_STREAM</td> </tr> </table> | MSG_OOB | Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only SOCK_STREAM |
| MSG_OOB | Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only SOCK_STREAM | | |

sockets created in the `AF_INET` address family support out-of-band data.

`MSG_DONTROUTE`

The `SO_DONTROUTE` option is turned on for the duration of the operation. It is used only by diagnostic or routing programs.

See `recv(3N)` for a description of the `msghdr` structure.

RETURN VALUES

These calls return the number of bytes sent, or `-1` if an error occurred.

ERRORS

The calls fail if:

EBADF

`s` is an invalid file descriptor.

EINTR

The operation was interrupted by delivery of a signal before any data could be buffered to be sent.

EINVAL

`toLen` is not the size of a valid address for the specified address family.

EMSGSIZE

The socket requires that message be sent atomically, and the message was too long.

ENOMEM

There was insufficient memory available to complete the operation.

ENOSR

There were insufficient STREAMS resources available for the operation to complete.

ENOTSOCK

`s` is not a socket.

EWouldBlock

The socket is marked non-blocking and the requested operation would block.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

`fcntl(2)`, `poll(2)`, `write(2)`, `connect(3N)`, `getsockopt(3N)`, `recv(3N)`, `select(3C)`, `socket(3N)`, `attributes(5)`, `socket(5)`

| | | | | | |
|--------------------|---|----------------|--|----------------|--|
| NAME | send – send a message on a socket | | | | |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lxnet [<i>library</i> ...] #include <sys/socket.h></pre> <pre>ssize_t send(int <i>socket</i>, const void *<i>buffer</i>, size_t <i>length</i>, int <i>flags</i>);</pre> | | | | |
| DESCRIPTION | <p><i>socket</i> Specifies the socket file descriptor.</p> <p><i>buffer</i> Points to the buffer containing the message to send.</p> <p><i>length</i> Specifies the length of the message in bytes.</p> <p><i>flags</i> Specifies the type of message transmission. Values of this argument are formed by logically OR'ing zero or more of the following flags:</p> <table border="0" style="margin-left: 2em;"> <tr> <td style="padding-right: 2em;">MSG_EOR</td> <td>Terminates a record (if supported by the protocol)</td> </tr> <tr> <td style="padding-right: 2em;">MSG_OOB</td> <td>Sends out-of-band data on sockets that support out-of-band communications. The significance and semantics of out-of-band data are protocol-specific.</td> </tr> </table> <p>The send() function initiates transmission of a message from the specified socket to its peer. The send() function sends a message only when the socket is connected (including when the peer of a connectionless socket has been set via connect(3XN)).</p> <p>The length of the message to be sent is specified by the <i>length</i> argument. If the message is too long to pass through the underlying protocol, send() fails and no data is transmitted.</p> <p>Successful completion of a call to send() does not guarantee delivery of the message. A return value of -1 indicates only locally-detected errors.</p> <p>If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does not have O_NONBLOCK set, send() blocks until space is available. If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does have O_NONBLOCK set, send() will fail. The select(3C) and poll(2) functions can be used to determine when it is possible to send more data.</p> | MSG_EOR | Terminates a record (if supported by the protocol) | MSG_OOB | Sends out-of-band data on sockets that support out-of-band communications. The significance and semantics of out-of-band data are protocol-specific. |
| MSG_EOR | Terminates a record (if supported by the protocol) | | | | |
| MSG_OOB | Sends out-of-band data on sockets that support out-of-band communications. The significance and semantics of out-of-band data are protocol-specific. | | | | |

| | | |
|----------------------|---|--|
| | The socket in use may require the process to have appropriate privileges to use the send() function. | |
| USAGE | The send() function is identical to sendto(3XN) with a null pointer <i>dest_len</i> argument, and to <i>write()</i> if no flags are used. | |
| RETURN VALUES | Upon successful completion, send() returns the number of bytes sent. Otherwise, -1 is returned and <i>errno</i> is set to indicate the error. | |
| ERRORS | The send() function will fail if: | |
| | EAGAIN | |
| | EWOULDBLOCK | The socket's file descriptor is marked <i>O_NONBLOCK</i> and the requested operation would block. |
| | EBADF | The <i>socket</i> argument is not a valid file descriptor. |
| | ECONNRESET | A connection was forcibly closed by a peer. |
| | EDESTADDRREQ | The socket is not connection-mode and no peer address is set. |
| | EFAULT | The <i>buffer</i> parameter can not be accessed. |
| | EINTR | A signal interrupted send() before any data was transmitted. |
| | EMSGSIZE | The message is too large be sent all at once, as the socket requires. |
| | ENOTCONN | The socket is not connected or otherwise has not had the peer prespecified. |
| | ENOTSOCK | The <i>socket</i> argument does not refer to a socket. |
| | EOPNOTSUPP | The <i>socket</i> argument is associated with a socket that does not support one or more of the values set in <i>flags</i> . |
| | EPIPE | The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type <i>SOCK_STREAM</i> , the <i>SIGPIPE</i> signal is generated to the calling process. |
| | The send() function may fail if: | |

| | |
|--------------------|--|
| EACCES | The calling process does not have the appropriate privileges. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| ENETDOWN | The local interface used to reach the destination is down. |
| ENETUNREACH | No route to the network is present. |
| ENOBUFS | Insufficient resources were available in the system to perform the operation. |
| ENOSR | There were insufficient STREAMS resources available for the operation to complete. |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`connect(3XN)`, `getsockopt(3XN)`, `poll(2)`, `recv(3XN)`, `recvfrom(3XN)`, `recvmsg(3XN)`, `select(3C)`, `sendmsg(3XN)`, `sendto(3XN)`, `setsockopt(3XN)`, `shutdown(3XN)`, `socket(3XN)`, `attributes(5)`

| | | | | | |
|--------------------|---|----------------|--|----------------|---|
| NAME | sendmsg – send a message on a socket using a message structure | | | | |
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lxnet [<i>library ...</i>] #include <sys/socket.h></pre> <p>ssize_t sendmsg(int <i>socket</i>, const struct msghdr *<i>message</i>, int <i>flags</i>);</p> | | | | |
| DESCRIPTION | <p>The sendmsg() function sends a message through a connection-mode or connectionless-mode socket. If the socket is connectionless-mode, the message will be sent to the address specified by <i>msghdr</i>. If the socket is connection-mode, the destination address in <i>msghdr</i> is ignored.</p> <p>The function takes the following arguments:</p> <p>socket Specifies the socket file descriptor.</p> <p>message Points to a <i>msghdr</i> structure, containing both the destination address and the buffers for the outgoing message. The length and format of the address depend on the address family of the socket. The <i>msg_flags</i> member is ignored.</p> <p>flags Specifies the type of message transmission. The application may specify 0 or the following flag:</p> <table border="0" style="margin-left: 2em;"> <tr> <td style="padding-right: 2em;">MSG_EOR</td> <td>Terminates a record (if supported by the protocol)</td> </tr> <tr> <td style="padding-right: 2em;">MSG_OOB</td> <td>Sends out-of-band data on sockets that support out-of-bound data. The significance and semantics of out-of-band data are protocol-specific.</td> </tr> </table> <p>The <i>msg_iov</i> and <i>msg_iovlen</i> fields of message specify zero or more buffers containing the data to be sent. <i>msg_iov</i> points to an array of <i>iovec</i> structures; <i>msg_iovlen</i> must be set to the dimension of this array. In each <i>iovec</i> structure, the <i>iov_base</i> field specifies a storage area and the <i>iov_len</i> field gives its size in bytes. Some of these sizes can be zero. The data from each storage area indicated by <i>msg_iov</i> is sent in turn.</p> <p>Successful completion of a call to sendmsg() does not guarantee delivery of the message. A return value of -1 indicates only locally-detected errors.</p> <p>If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does not have O_NONBLOCK set, sendmsg() function blocks until space is available. If space is not available at</p> | MSG_EOR | Terminates a record (if supported by the protocol) | MSG_OOB | Sends out-of-band data on sockets that support out-of-bound data. The significance and semantics of out-of-band data are protocol-specific. |
| MSG_EOR | Terminates a record (if supported by the protocol) | | | | |
| MSG_OOB | Sends out-of-band data on sockets that support out-of-bound data. The significance and semantics of out-of-band data are protocol-specific. | | | | |

the sending socket to hold the message to be transmitted and the socket file descriptor does have O_NONBLOCK set, **sendmsg()** function will fail.

If the socket protocol supports broadcast and the specified address is a broadcast address for the socket protocol, **sendmsg()** will fail if the SO_BROADCAST option is not set for the socket.

The socket in use may require the process to have appropriate privileges to use the **sendmsg()** function.

USAGE

The **select(3C)** and **poll(2)** functions can be used to determine when it is possible to send more data.

RETURN VALUES

Upon successful completion, **sendmsg()** function returns the number of bytes sent. Otherwise, -1 is returned and **errno** is set to indicate the error.

ERRORS

The **sendmsg()** function will fail if:

EAGAIN**EWOULDBLOCK**

The socket's file descriptor is marked O_NONBLOCK and the requested operation would block.

EAFNOSUPPORT

Addresses in the specified address family cannot be used with this socket.

EBADF

The *socket* argument is not a valid file descriptor.

ECONNRESET

A connection was forcibly closed by a peer.

EFAULT

The *message* parameter, or storage pointed to by the *msg_name*, *msg_control* or *msg_iov* fields of the *message* parameter, or storage pointed to by the *iovec* structures pointed to by the *msg_iov* field can not be accessed.

EINTR

A signal interrupted **sendmsg()** before any data was transmitted.

EINVAL

The sum of the *iov_len* values overflows an *ssize_t*.

EMSGSIZE

The message is too large to be sent all at once (as the socket requires), or the *msg_iovlen* member of the *msghdr* structure pointed to by *message* is less than or equal to 0 or is greater than IOV_MAX.

| | |
|---|--|
| ENOTCONN | The socket is connection-mode but is not connected. |
| ENOTSOCK | The <i>socket</i> argument does not refer a socket. |
| EOPNOTSUPP | The <i>socket</i> argument is associated with a socket that does not support one or more of the values set in <i>flags</i> . |
| EPIPE | The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type <code>SOCK_STREAM</code> , the <code>SIGPIPE</code> signal is generated to the calling process. |
| If the address family of the socket is <code>AF_UNIX</code> , then sendmsg() will fail if: | |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| ELOOP | Too many symbolic links were encountered in translating the pathname in the socket address. |
| ENAMETOOLONG | A component of a pathname exceeded <code>NAME_MAX</code> characters, or an entire pathname exceeded <code>PATH_MAX</code> characters. |
| ENOENT | A component of the pathname does not name an existing file or the pathname is an empty string. |
| ENOTDIR | A component of the path prefix of the pathname in the socket address is not a directory. |
| The sendmsg() function may fail if: | |
| EACCES | Search permission is denied for a component of the path prefix; or write access to the named socket is denied. |
| EDESTADDRREQ | The socket is not connection-mode and does not have its peer address set, and no destination address was specified. |
| EHOSTUNREACH | The destination host cannot be reached (probably because the host is down or a remote router cannot reach it). |
| EIO | An I/O error occurred while reading from or writing to the file system. |

| | |
|--------------------|--|
| EISCONN | A destination address was specified and the socket is already connected. |
| ENETDOWN | The local interface used to reach the destination is down. |
| ENETUNREACH | No route to the network is present. |
| ENOBUFS | Insufficient resources were available in the system to perform the operation. |
| ENOMEM | Insufficient memory was available to fulfill the request. |
| ENOSR | There were insufficient STREAMS resources available for the operation to complete. |

If the address family of the socket is AF_UNIX, then **sendmsg()** may fail if:

| | |
|---------------------|---|
| ENAMETOOLONG | Pathname resolution of a symbolic link produced an intermediate result whose length exceeds PATH_MAX. |
|---------------------|---|

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

poll(2), **getsockopt(3XN)**, **recv(3XN)**, **recvfrom(3XN)**, **recvmsg(3XN)**, **select(3C)**, **send(3XN)**, **sendto(3XN)**, **setsockopt(3XN)**, **shutdown(3XN)**, **socket(3XN)**, **attributes(5)**

| | | | | | |
|--------------------|---|----------------|--|----------------|--|
| NAME | sendto – send a message on a socket | | | | |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lxnet [<i>library</i> ...] #include <sys/socket.h></pre> <p> <code>ssize_t sendto(int socket, const void *message, size_t length, int flags, const struct sockaddr *dest_addr, socklen_t dest_len);</code> </p> | | | | |
| DESCRIPTION | <p>The sendto() function sends a message through a connection-mode or connectionless-mode socket. If the socket is connectionless-mode, the message will be sent to the address specified by <i>dest_addr</i>. If the socket is connection-mode, <i>dest_addr</i> is ignored.</p> <p>The function takes the following arguments:</p> <p>socket Specifies the socket file descriptor.</p> <p>message Points to a buffer containing the message to be sent.</p> <p>length Specifies the size of the message in bytes.</p> <p>flags Specifies the type of message transmission. Values of this argument are formed by logically OR'ing zero or more of the following flags:</p> <table border="0" style="margin-left: 2em;"> <tr> <td style="padding-right: 2em;">MSG_EOR</td> <td>Terminates a record (if supported by the protocol)</td> </tr> <tr> <td style="padding-right: 2em;">MSG_OOB</td> <td>Sends out-of-band data on sockets that support out-of-band data. The significance and semantics of out-of-band data are protocol specific.</td> </tr> </table> <p>dest_addr Points to a <code>sockaddr</code> structure containing the destination address. The length and format of the address depend on the address family of the socket.</p> <p>dest_len Specifies the length of the <code>sockaddr</code> structure pointed to by the <i>dest_addr</i> argument.</p> <p>If the socket protocol supports broadcast and the specified address is a broadcast address for the socket protocol, sendto() will fail if the <code>SO_BROADCAST</code> option is not set for the socket.</p> <p>The <i>dest_addr</i> argument specifies the address of the target. The <i>length</i> argument specifies the length of the message.</p> | MSG_EOR | Terminates a record (if supported by the protocol) | MSG_OOB | Sends out-of-band data on sockets that support out-of-band data. The significance and semantics of out-of-band data are protocol specific. |
| MSG_EOR | Terminates a record (if supported by the protocol) | | | | |
| MSG_OOB | Sends out-of-band data on sockets that support out-of-band data. The significance and semantics of out-of-band data are protocol specific. | | | | |

| | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------|---|---------------------|--|---------------|--|--------------------|---|--------------|--|-------------------|---|---------------|--|--------------|---|-----------------|--|-----------------|---|-----------------|--|-------------------|--|
| | <p>Successful completion of a call to sendto() does not guarantee delivery of the message. A return value of <code>-1</code> indicates only locally-detected errors.</p> <p>If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does not have <code>O_NONBLOCK</code> set, sendto() blocks until space is available. If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does have <code>O_NONBLOCK</code> set, sendto() will fail.</p> <p>The socket in use may require the process to have appropriate privileges to use the sendto() function.</p> | | | | | | | | | | | | | | | | | | | | | | |
| USAGE | <p>The <code>select(3C)</code> and <code>poll(2)</code> functions can be used to determine when it is possible to send more data.</p> | | | | | | | | | | | | | | | | | | | | | | |
| RETURN VALUES | <p>Upon successful completion, sendto() returns the number of bytes sent. Otherwise, <code>-1</code> is returned and <code>errno</code> is set to indicate the error.</p> | | | | | | | | | | | | | | | | | | | | | | |
| ERRORS | <p>The sendto() function will fail if:</p> <table border="0"> <tr> <td>EAFNOSUPPORT</td> <td>Addresses in the specified address family cannot be used with this socket.</td> </tr> <tr> <td>EAGAIN</td> <td></td> </tr> <tr> <td>EWOULDBLOCK</td> <td>The socket's file descriptor is marked <code>O_NONBLOCK</code> and the requested operation would block.</td> </tr> <tr> <td>EBADF</td> <td>The <i>socket</i> argument is not a valid file descriptor.</td> </tr> <tr> <td>ECONNRESET</td> <td>A connection was forcibly closed by a peer.</td> </tr> <tr> <td>EFAULT</td> <td>The <i>message</i> or <i>destaddr</i> parameter can not be accessed.</td> </tr> <tr> <td>EINTR</td> <td>A signal interrupted sendto() before any data was transmitted.</td> </tr> <tr> <td>EMSGSIZE</td> <td>The message is too large to be sent all at once, as the socket requires.</td> </tr> <tr> <td>ENOTCONN</td> <td>The socket is connection-mode but is not connected.</td> </tr> <tr> <td>ENOTSOCK</td> <td>The <i>socket</i> argument does not refer to a socket.</td> </tr> <tr> <td>EOPNOTSUPP</td> <td>The <i>socket</i> argument is associated with a socket that does not support one or more of the values set in <i>flags</i>.</td> </tr> </table> | EAFNOSUPPORT | Addresses in the specified address family cannot be used with this socket. | EAGAIN | | EWOULDBLOCK | The socket's file descriptor is marked <code>O_NONBLOCK</code> and the requested operation would block. | EBADF | The <i>socket</i> argument is not a valid file descriptor. | ECONNRESET | A connection was forcibly closed by a peer. | EFAULT | The <i>message</i> or <i>destaddr</i> parameter can not be accessed. | EINTR | A signal interrupted sendto() before any data was transmitted. | EMSGSIZE | The message is too large to be sent all at once, as the socket requires. | ENOTCONN | The socket is connection-mode but is not connected. | ENOTSOCK | The <i>socket</i> argument does not refer to a socket. | EOPNOTSUPP | The <i>socket</i> argument is associated with a socket that does not support one or more of the values set in <i>flags</i> . |
| EAFNOSUPPORT | Addresses in the specified address family cannot be used with this socket. | | | | | | | | | | | | | | | | | | | | | | |
| EAGAIN | | | | | | | | | | | | | | | | | | | | | | | |
| EWOULDBLOCK | The socket's file descriptor is marked <code>O_NONBLOCK</code> and the requested operation would block. | | | | | | | | | | | | | | | | | | | | | | |
| EBADF | The <i>socket</i> argument is not a valid file descriptor. | | | | | | | | | | | | | | | | | | | | | | |
| ECONNRESET | A connection was forcibly closed by a peer. | | | | | | | | | | | | | | | | | | | | | | |
| EFAULT | The <i>message</i> or <i>destaddr</i> parameter can not be accessed. | | | | | | | | | | | | | | | | | | | | | | |
| EINTR | A signal interrupted sendto() before any data was transmitted. | | | | | | | | | | | | | | | | | | | | | | |
| EMSGSIZE | The message is too large to be sent all at once, as the socket requires. | | | | | | | | | | | | | | | | | | | | | | |
| ENOTCONN | The socket is connection-mode but is not connected. | | | | | | | | | | | | | | | | | | | | | | |
| ENOTSOCK | The <i>socket</i> argument does not refer to a socket. | | | | | | | | | | | | | | | | | | | | | | |
| EOPNOTSUPP | The <i>socket</i> argument is associated with a socket that does not support one or more of the values set in <i>flags</i> . | | | | | | | | | | | | | | | | | | | | | | |

| | |
|--|--|
| EPIPE | The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type <code>SOCK_STREAM</code> , the <code>SIGPIPE</code> signal is generated to the calling process. |
| If the address family of the socket is <code>AF_UNIX</code> , then sendto() will fail if: | |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| ELOOP | Too many symbolic links were encountered in translating the pathname in the socket address. |
| ENAMETOOLONG | A component of a pathname exceeded <code>NAME_MAX</code> characters, or an entire pathname exceeded <code>PATH_MAX</code> characters. |
| ENOENT | A component of the pathname does not name an existing file or the pathname is an empty string. |
| ENOTDIR | A component of the path prefix of the pathname in the socket address is not a directory. |
| The sendto() function may fail if: | |
| EACCES | Search permission is denied for a component of the path prefix; or write access to the named socket is denied. |
| EDESTADDRREQ | The socket is not connection-mode and does not have its peer address set, and no destination address was specified. |
| EHOSTUNREACH | The destination host cannot be reached (probably because the host is down or a remote router cannot reach it). |
| EINVAL | The <i>dest_len</i> argument is not a valid length for the address family. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| EISCONN | A destination address was specified and the socket is already connected. |
| ENETDOWN | The local interface used to reach the destination is down. |

ENETUNREACH No route to the network is present.

ENOBUFS Insufficient resources were available in the system to perform the operation.

ENOMEM Insufficient memory was available to fulfill the request.

ENOSR There were insufficient STREAMS resources available for the operation to complete.

If the address family of the socket is AF_UNIX, then **sendto()** may fail if:

ENAMETOOLONG Pathname resolution of a symbolic link produced an intermediate result whose length exceeds PATH_MAX.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

poll(2), **getsockopt(3XN)**, **recv(3XN)**, **recvfrom(3XN)**, **recvmsg(3XN)**, **select(3C)**, **send(3XN)**, **sendmsg(3XN)**, **setsockopt(3XN)**, **shutdown(3XN)**, **socket(3XN)**, **attributes(5)**

| | |
|----------------------|--|
| NAME | setbuf, setvbuf – assign buffering to a stream |
| SYNOPSIS | <pre>#include <stdio.h> void setbuf(FILE * stream, char * buf); int setvbuf(FILE * stream, char * buf, int type, size_t size);</pre> |
| DESCRIPTION | <p>The setbuf() function may be used after the stream pointed to by <i>stream</i> (see intro(3)) is opened but before it is read or written. It causes the array pointed to by <i>buf</i> to be used instead of an automatically allocated buffer. If <i>buf</i> is the null pointer, input/output will be completely unbuffered. The constant <code>BUFSIZ</code>, defined in the <code><stdio.h></code> header, indicates the size of the array pointed to by <i>buf</i>.</p> <p>The setvbuf() function may be used after a stream is opened but before it is read or written. The <i>type</i> argument determines how <i>stream</i> will be buffered. Legal values for <i>type</i> (defined in <code><stdio.h></code>) are:</p> <p><code>_IOFBF</code> Input/output to be fully buffered.</p> <p><code>_IOLBF</code> Output to be line buffered; the buffer will be flushed when a <code>NEWLINE</code> is written, the buffer is full, or input is requested.</p> <p><code>_IONBF</code> Input/output to be completely unbuffered.</p> <p>If <i>buf</i> is not the null pointer, the array it points to will be used for buffering, instead of an automatically allocated buffer. The <i>size</i> argument specifies the size of the buffer to be used. If input/output is unbuffered, <i>buf</i> and <i>size</i> are ignored.</p> <p>For a further discussion of buffering, see stdio(3S).</p> |
| RETURN VALUES | If an illegal value for <i>type</i> is provided, setvbuf() returns a non-zero value. Otherwise, it returns 0. |
| USAGE | <p>A common source of error is allocating buffer space as an “automatic” variable in a code block, and then failing to close the stream in the same block.</p> <p>When using setbuf(), <i>buf</i> should always be sized using <code>BUFSIZ</code>. If the array pointed to by <i>buf</i> is larger than <code>BUFSIZ</code>, a portion of <i>buf</i> will not be used. If <i>buf</i> is smaller than <code>BUFSIZ</code>, other memory may be unexpectedly overwritten.</p> <p>Parts of <i>buf</i> will be used for internal bookkeeping of the stream and, therefore, <i>buf</i> will contain less than <i>size</i> bytes when full. It is recommended that stdio(3S) be used to handle buffer allocation when using setvbuf().</p> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`fopen(3S)` , `getc(3S)` , `malloc(3C)` , `putc(3S)` , `stdio(3S)` ,
`attributes(5)`

| | |
|----------------------|---|
| NAME | setbuffer, setlinebuf – assign buffering to a stream |
| SYNOPSIS | <pre>#include <stdio.h> void setbuffer(FILE * <i>iop</i>, char * <i>abuf</i>, size_t <i>asize</i>); int setlinebuf(FILE * <i>iop</i>);</pre> |
| DESCRIPTION | <p>The setbuffer() and setlinebuf() functions assign buffering to a stream. The three types of buffering available are unbuffered, block buffered, and line buffered. When an output stream is unbuffered, information appears on the destination file or terminal as soon as written; when it is block buffered, many characters are saved and written as a block; when it is line buffered, characters are saved until either a NEWLINE is encountered or input is read from <code>stdin</code>. The fflush(3S) function may be used to force the block out early. Normally all files are block buffered. A buffer is obtained from malloc(3C) upon the first getc(3S) or putc(3S) performed on the file. If the standard stream <code>stdout</code> refers to a terminal, it is line buffered. The standard stream <code>stderr</code> is unbuffered by default.</p> <p>The setbuffer() function can be used after a stream <i>iop</i> has been opened but before it is read or written. It uses the character array <i>abuf</i> whose size is determined by the <i>asize</i> argument instead of an automatically allocated buffer. If <i>abuf</i> is the null pointer, input/output will be completely unbuffered. A manifest constant <code>BUFSIZ</code>, defined in the <code><stdio.h></code> header, tells how large an array is needed:</p> <pre>char buf[BUFSIZ];</pre> <p>The setlinebuf() function is used to change the buffering on a stream from block buffered or unbuffered to line buffered. Unlike setbuffer(), it can be used at any time that the stream <i>iop</i> is active.</p> <p>A stream can be changed from unbuffered or line buffered to block buffered by using freopen(3S). A stream can be changed from block buffered or line buffered to unbuffered by using freopen(3S) followed by setbuf(3S) with a buffer argument of <code>NULL</code>.</p> |
| RETURN VALUES | The setlinebuf() function returns no useful value. |
| SEE ALSO | malloc(3C) , fclose(3S) , fopen(3S) , fread(3S) , getc(3S) , printf(3S) , putc(3S) , puts(3S) , setbuf(3S) , setvbuf(3S) |

NOTES

A common source of error is allocating buffer space as an “automatic” variable in a code block, and then failing to close the stream in the same block.

| NAME | setcat – define default catalog | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <pfmt.h> char *setcat(const char *catalog);</pre> | | | | |
| DESCRIPTION | <p>The setcat() function defines the default message catalog to be used by subsequent calls to gettext(3C), lfmt(3C), or pfmt(3C) that do not explicitly specify a message catalog.</p> <p>The <i>catalog</i> argument must be limited to 14 characters. These characters must be selected from a set of all characters values, excluding \0 (null) and the ASCII codes for / (slash) and : (colon).</p> <p>The setcat() function assumes that the catalog exists. No checking is done on the argument.</p> <p>A null pointer passed as an argument will result in the return of a pointer to the current default message catalog name. A pointer to an empty string passed as an argument will cancel the default catalog.</p> <p>If no default catalog is specified, or if <i>catalog</i> is an invalid catalog name, subsequent calls to gettext(3C), lfmt(3C), or pfmt(3C) that do not explicitly specify a catalog name will use Message not found!\n as default string.</p> | | | | |
| RETURN VALUES | Upon successful completion, setcat() returns a pointer to the catalog name. Otherwise, it returns a null pointer. | | | | |
| EXAMPLES | <p>EXAMPLE 1 Example of setcat() function.</p> <pre>setcat("test"); gettext(":10", "hello world\n")</pre> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | gettext(3C) , lfmt(3C) , pfmt(3C) , setlocale(3C) , attributes(5) , environ(5) | | | | |

| | | | | | | | | | | | |
|----------------------|--|-------------|--|------------|-----------------------------------|--------------|---|-------------------|--|-------------|---|
| NAME | setcchar – set a <code>cchar_t</code> type character from a wide character and rendition | | | | | | | | | | |
| SYNOPSIS | <pre>#include <curses.h> int setcchar(cchar_t *wcvl, const wchar_t *wch, const attr_t attrs, short color_pair, const void *opts);</pre> | | | | | | | | | | |
| PARAMETERS | <table><tr><td>wcvl</td><td>Is a pointer to a location where a <code>cchar_t</code> character (and its rendition) can be stored.</td></tr><tr><td>wch</td><td>Is a pointer to a wide character.</td></tr><tr><td>attrs</td><td>Is the set of attributes to apply to <i>wch</i> in creating <i>wcvl</i>.</td></tr><tr><td>color_pair</td><td>Is the color pair to apply to <i>wch</i> in creating <i>wcvl</i>.</td></tr><tr><td>opts</td><td>Is reserved for future use. Currently, this must be a null pointer.</td></tr></table> | wcvl | Is a pointer to a location where a <code>cchar_t</code> character (and its rendition) can be stored. | wch | Is a pointer to a wide character. | attrs | Is the set of attributes to apply to <i>wch</i> in creating <i>wcvl</i> . | color_pair | Is the color pair to apply to <i>wch</i> in creating <i>wcvl</i> . | opts | Is reserved for future use. Currently, this must be a null pointer. |
| wcvl | Is a pointer to a location where a <code>cchar_t</code> character (and its rendition) can be stored. | | | | | | | | | | |
| wch | Is a pointer to a wide character. | | | | | | | | | | |
| attrs | Is the set of attributes to apply to <i>wch</i> in creating <i>wcvl</i> . | | | | | | | | | | |
| color_pair | Is the color pair to apply to <i>wch</i> in creating <i>wcvl</i> . | | | | | | | | | | |
| opts | Is reserved for future use. Currently, this must be a null pointer. | | | | | | | | | | |
| DESCRIPTION | The <code>setcchar()</code> function takes the wide character pointed to by <i>wch</i> , combines it with the attributes indicated by <i>attrs</i> and the color pair indicated by <i>color_pair</i> and stores the result in the object pointed to by <i>wcvl</i> . | | | | | | | | | | |
| RETURN VALUES | On success, the <code>setcchar()</code> function returns <code>OK</code> . Otherwise, it returns <code>ERR</code> . | | | | | | | | | | |
| ERRORS | None. | | | | | | | | | | |
| SEE ALSO | <code>attroff(3XC)</code> , <code>can_change_color(3XC)</code> , <code>getcchar(3XC)</code> | | | | | | | | | | |

| | |
|--------------------|--|
| NAME | setjmp, longjmp, _setjmp, _longjmp – non-local goto |
| SYNOPSIS | <pre> /usr/ucb/cc [flag ...] file ... #include <setjmp.h> int setjmp(env); jmp_buf env ; void longjmp(env, val); jmp_buf env ; int val ; int _setjmp(env); jmp_buf env ; void _longjmp(env, val); jmp_buf env ; int val ; </pre> |
| DESCRIPTION | <p>setjmp() and longjmp() are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.</p> <p>setjmp() saves its stack environment in <i>env</i> for later use by longjmp(). A normal call to setjmp() returns zero. setjmp() also saves the register environment. If a longjmp() call will be made, the routine which called setjmp() should not return until after the longjmp() has returned control (see below).</p> |

longjmp() restores the environment saved by the last call of **setjmp()**, and then returns in such a way that execution continues as if the call of **setjmp()** had just returned the value *val* to the function that invoked **setjmp()**; however, if *val* were zero, execution would continue as if the call of **setjmp()** had returned one. This ensures that a "return" from **setjmp()** caused by a call to **longjmp()** can be distinguished from a regular return from **setjmp()**. The calling function must not itself have returned in the interim, otherwise **longjmp()** will be returning control to a possibly non-existent environment. All memory-bound data have values as of the time **longjmp()** was called. The CPU and floating-point data registers are restored to the values they had at the time that **setjmp()** was called. But, because the `register` storage class is only a hint to the C compiler, variables declared as `register` variables may not necessarily be assigned to machine registers, so their values are unpredictable after a **longjmp()**. This is especially a problem for programmers trying to write machine-independent C routines.

setjmp() and **longjmp()** save and restore the signal mask while **_setjmp()** and **_longjmp()** manipulate only the C stack and registers.

None of these functions save or restore any floating-point status or control registers.

EXAMPLES

EXAMPLE 1 Examples of **setjmp()** and **longjmp()**.

The following example uses both **setjmp()** and **longjmp()** to return the flow of control to the appropriate instruction block:

```
#include <stdio.h>
#include <setjmp.h>
#include <signal.h>
#include <unistd.h>
jmp_buf env; static void signal_handler();
main() {
    int returned_from_longjump, processing = 1;
    unsigned int time_interval = 4;
    if ((returned_from_longjump = setjmp(env)) != 0)
        switch (returned_from_longjump) {
            case SIGINT:
                printf("longjumped from interrupt %d\
", SIGINT);
                break;
            case SIGALRM:
                printf("longjumped from alarm %d\
", SIGALRM);
                break;
        }
    (void) signal(SIGINT, signal_handler);
    (void) signal(SIGALRM, signal_handler);
    alarm(time_interval);
    while (processing) {
        printf(" waiting for you to INTERRUPT (cntrl-C) ...\
");
        sleep(1);
    }
}
```

```

        }\011\011
    /* end while forever loop */
}

static void signal_handler(sig)
int sig; {
    switch (sig) {
        case SIGINT:\011    ... \011
    /* process for interrupt */

                                \011    longjmp(env, sig);

    /* break never reached */

        case SIGALRM:    ... \011
    /* process for alarm */

                                longjmp(env, sig);
                                \011\011

        /* break never reached */

        default:
    exit(sig);
    }
}

```

When this example is compiled and executed, and the user sends an interrupt signal, the output will be:

```
longjumped from interrupt
```

Additionally, every 4 seconds the alarm will expire, signalling this process, and the output will be:

```
longjumped from alarm
```

SEE ALSO `cc(1B)`, `sigvec(3B)`, `setjmp(3C)`, `signal(3C)`

NOTES Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

BUGS `setjmp()` does not save the current notion of whether the process is executing on the signal stack. The result is that a `longjmp()` to some place on the signal stack leaves the signal stack state incorrect.

On some systems **setjmp()** also saves the register environment. Therefore, all data that are bound to registers are restored to the values they had at the time that **setjmp()** was called. All memory-bound data have values as of the time **longjmp()** was called. However, because the `register` storage class is only a hint to the C compiler, variables declared as `register` variables may not necessarily be assigned to machine registers, so their values are unpredictable after a **longjmp()**. When using compiler options that specify automatic register allocation (see `cc(1B)`), the compiler will not attempt to assign variables to registers in routines that call **setjmp()**.

longjmp() never causes **setjmp()** to return zero, so programmers should not depend on **longjmp()** being able to cause **setjmp()** to return zero.

| | |
|--------------------|--|
| NAME | setjmp, sigsetjmp, longjmp, siglongjmp – non-local goto |
| SYNOPSIS | <pre>#include <setjmp.h> int setjmp(jmp_buf env); int sigsetjmp(sigjmp_buf env, int savemask); void longjmp(jmp_buf env, int val); void siglongjmp(sigjmp_buf env, int val);</pre> |
| DESCRIPTION | <p>These functions are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.</p> <p>The setjmp() function saves its stack environment in <i>env</i> for later use by longjmp() .</p> <p>The sigsetjmp() function saves the calling process's registers and stack environment (see sigaltstack(2)) in <i>env</i> for later use by siglongjmp() . If <i>savemask</i> is non-zero, the calling process's signal mask (see sigprocmask(2)) and scheduling parameters (see pricntl(2)) are also saved.</p> <p>The longjmp() function restores the environment saved by the last call of setjmp() with the corresponding <i>env</i> argument. After longjmp() completes, program execution continues as if the corresponding call to setjmp() had just returned the value <i>val</i> . The caller of setjmp() must not have returned in the interim. The longjmp() function cannot cause setjmp() to return the value 0. If longjmp() is invoked with a second argument of 0, setjmp() will return 1. At the time of the second return from setjmp() , all external and static variables have values as of the time longjmp() is called (see EXAMPLES) .</p> <p>The siglongjmp() function restores the environment saved by the last call of sigsetjmp() with the corresponding <i>env</i> argument. After siglongjmp() completes, program execution continues as if the corresponding call to sigsetjmp() had just returned the value <i>val</i> . The siglongjmp() function cannot cause sigsetjmp() to return the value 0. If siglongjmp() is invoked with a second argument of 0, sigsetjmp() will return 1. At the time of the second return from sigsetjmp() , all external and static variables have values as of the time siglongjmp() was called.</p> <p>If a signal-catching function interrupts sleep(3C) and calls siglongjmp() to restore an environment saved prior to the sleep() call, the action associated with SIGALRM and time it is scheduled to be generated are unspecified. It is also unspecified whether the SIGALRM signal is blocked, unless the process's signal mask is restored as part of the environment.</p> |

The **siglongjmp()** function restores the saved signal mask if and only if the *env* argument was initialized by a call to the **sigsetjmp()** function with a non-zero *savemask* argument.

The values of register and automatic variables are undefined. Register or automatic variables whose value must be relied upon must be declared as volatile.

RETURN VALUES

If **longjmp()** or **siglongjmp()** are invoked with a second argument of 0, **setjmp()** and **sigsetjmp()**, respectively, return 1. Otherwise, **setjmp()** and **sigsetjmp()** return 0.

EXAMPLES

EXAMPLE 1 Example of **setjmp()** and **longjmp()** functions.

The following example uses both **setjmp()** and **longjmp()** to return the flow of control to the appropriate instruction block:

```
#include <stdio.h>
#include <setjmp.h>
#include <signal.h>
#include <unistd.h>
jmp_buf env; static void signal_handler();

main() {
    int returned_from_longjump, processing = 1;
    unsigned int time_interval = 4;
    if ((returned_from_longjump = setjmp(env)) != 0)
        switch (returned_from_longjump) {
            case SIGINT:
                printf("longjumped from interrupt %d\
", SIGINT);
                break;
            case SIGALRM:
                printf("longjumped from alarm %d\
", SIGALRM);
                break;
        }
    (void) signal(SIGINT, signal_handler);
    (void) signal(SIGALRM, signal_handler);
    alarm(time_interval);
    while (processing) {
        printf(" waiting for you to INTERRUPT (cntrl-C) ...\
");
        sleep(1);
    }\011\011
    /* end while forever loop */
}

static void signal_handler(sig)
int sig; {
    switch (sig) {
        case SIGINT:\011... \011
```

```

/* process for interrupt */

        \011longjmp(env,sig);
        \011\011
/* break never reached */

        case SIGALRM:\011... \011
/* process for alarm */

        \011longjmp(env,sig);
        \011\011
/* break never reached */

        default:    \011exit(sig);
    }
}

```

When this example is compiled and executed, and the user sends an interrupt signal, the output will be:

```
longjumped from interrupt
```

Additionally, every 4 seconds the alarm will expire, signalling this process, and the output will be:

```
longjumped from alarm
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

getcontext(2), **pricntl(2)**, **sigaction(2)**, **sigaltstack(2)**, **sigprocmask(2)**, **signal(3C)**, **attributes(5)**

WARNINGS

If **longjmp()** or **siglongjmp()** are called even though *env* was never primed by a call to **setjmp()** or **sigsetjmp()**, or when the last such call was in a function that has since returned, the results are undefined.

| NAME | setkey – set encoding key | | | | |
|----------------------|---|----------------|-----------------|----------|------|
| SYNOPSIS | #include <stdlib.h> void setkey (const char *key); | | | | |
| DESCRIPTION | The setkey() function provides (rather primitive) access to the hashing algorithm employed by the crypt(3C) function. The argument of setkey() is an array of length 64 bytes containing only the bytes with numerical value of 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored; this gives a 56-bit key which is used by the algorithm. This is the key that will be used with the algorithm to encode a string <i>block</i> passed to encrypt(3C) . | | | | |
| RETURN VALUES | No values are returned. | | | | |
| ERRORS | The setkey() function will fail if: ENOSYS The functionality is not supported on this implementation. | | | | |
| USAGE | In some environments, decoding may not be implemented. This is related to U.S. Government restrictions on encryption and decryption routines: the DES decryption algorithm cannot be exported outside the U.S.A. Historical practice has been to ship a different version of the encryption library without the decryption feature in the routines supplied. Thus the exported version of encrypt() does encoding but not decoding. Because setkey() does not return a value, applications wishing to check for errors should set errno to 0, call setkey() , then test errno and, if it is non-zero, assume an error has occurred. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: <table border="1" data-bbox="500 1346 1398 1432"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe | | | | |
| SEE ALSO | crypt(3C) , encrypt(3C) , attributes(5) | | | | |

NAME setlabel – define the label for **pfmt()** and **lfmt()**

SYNOPSIS `#include <pfmt.h>`
`int setlabel(const char *label);`

DESCRIPTION The **setlabel()** function defines the label for messages produced in standard format by subsequent calls to **lfmt(3C)** and **pfmt(3C)**.
 The *label* argument is a character string no more than 25 characters in length.
 No label is defined before **setlabel()** is called. The label should be set once at the beginning of a utility and remain constant. A null pointer or an empty string passed as argument will reset the definition of the label.

RETURN VALUE Upon successful completion, **setlabel()** returns 0; otherwise, it returns a non-zero value.

EXAMPLES The following code (without previous call to **setlabel()**):

```
pfmt(stderr, MM_ERROR, "test:2:Cannot open file\n");
setlabel("UX:test");
pfmt(stderr, MM_ERROR, "test:2:Cannot open file\n");
```

 will produce the following output:

```
ERROR: Cannot open file
UX:test: ERROR: Cannot open file
```

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **getopt(3C)**, **lfmt(3C)**, **pfmt(3C)**, **attributes(5)**

| | |
|--------------------|---|
| NAME | setlocale – modify and query a program's locale |
| SYNOPSIS | <pre>#include <locale.h> char *setlocale(int category, const char *locale);</pre> |
| DESCRIPTION | <p>The setlocale() function selects the appropriate piece of the program's locale as specified by the <i>category</i> and <i>locale</i> arguments. The <i>category</i> argument may have the following values: LC_CTYPE, LC_NUMERIC, LC_TIME, LC_COLLATE, LC_MONETARY, LC_MESSAGES, and LC_ALL. These names are defined in the <code><locale.h></code> header. The LC_ALL variable names all of a program's locale categories.</p> <p>The LC_CTYPE variable affects the behavior of character handling functions such as <code>isdigit(3C)</code> and <code>tolower(3C)</code>, and multibyte character functions such as <code>mbtowc(3C)</code> and <code>wctomb(3C)</code>.</p> <p>The LC_NUMERIC variable affects the decimal point character and thousands separator character for the formatted input/output functions and string conversion functions.</p> <p>The LC_TIME variable affects the date and time format as delivered by <code>ascftime(3C)</code>, <code>cftime(3C)</code>, <code>getdate(3C)</code>, <code>strftime(3C)</code> and <code>strptime(3C)</code>.</p> <p>The LC_COLLATE variable affects the sort order produced by collating functions such as <code>strcoll(3C)</code> and <code>strxfrm(3C)</code>.</p> <p>The LC_MONETARY variable affects the monetary formatted information returned by <code>localeconv(3C)</code>.</p> <p>The LC_MESSAGES variable affects the behavior of messaging functions such as <code>dgettext(3C)</code>, <code>gettext(3C)</code>, and <code>gettext(3C)</code>.</p> <p>A value of "C" for <i>locale</i> specifies the traditional UNIX system behavior. At program startup, the equivalent of</p> <pre>setlocale(LC_ALL, "C")</pre> <p>is executed. This has the effect of initializing each category to the locale described by the environment "C".</p> <p>A value of "" for <i>locale</i> specifies that the locale should be taken from environment variables. The order in which the environment variables are checked for the various categories is given below:</p> |

| Category | 1st Env Var | 2nd Env Var | 3rd Env Var |
|--------------|-------------|-------------|-------------|
| LC_CTYPE: | LC_ALL | LC_CTYPE | LANG |
| LC_COLLATE: | LC_ALL | LC_COLLATE | LANG |
| LC_CTIME: | LC_ALL | LC_CTIME | LANG |
| LC_NUMERIC: | LC_ALL | LC_NUMERIC | LANG |
| LC_MONETARY: | LC_ALL | LC_MONETARY | LANG |
| LC_MESSAGES: | LC_ALL | LC_MESSAGES | LANG |

If a pointer to a string is given for *locale*, **setlocale()** attempts to set the locale for the given category to *locale*. If **setlocale()** succeeds, *locale* is returned. If **setlocale()** fails, a null pointer is returned and the program's locale is not changed.

For category LC_ALL, the behavior is slightly different. If a pointer to a string is given for *locale* and LC_ALL is given for *category*, **setlocale()** attempts to set the locale for all the categories to *locale*. The *locale* may be a simple locale, consisting of a single locale, or a composite locale. If the locales for all the categories are the same after all the attempted locale changes, **setlocale()** will return a pointer to the common simple locale. If there is a mixture of locales among the categories, **setlocale()** will return a composite locale.

RETURN VALUES

Upon successful completion, **setlocale()** returns the string associated with the specified category for the new locale. Otherwise, **setlocale()** returns a null pointer and the program's locale is not changed.

A null pointer for *locale* causes **setlocale()** to return a pointer to the string associated with the *category* for the program's current locale. The program's locale is not changed.

The string returned by **setlocale()** is such that a subsequent call with that string and its associated *category* will restore that part of the program's locale. The string returned must not be modified by the program, but may be overwritten by a subsequent call to **setlocale()**.

ERRORS

No errors are defined.

FILES

`/usr/lib/locale/locale` locale database directory for *locale*

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |
| CSI | Enabled |

SEE ALSO

`locale(1)`, `ctype(3C)`, `getdate(3C)` `gettext(3C)`, `gettext(3C)`, `isdigit(3C)`, `localeconv(3C)`, `mbtowc(3C)`, `strcoll(3C)`, `strftime(3C)`, `strptime(3C)` `strxfrm(3C)` `tolower(3C)`, `wctomb(3C)`, `libc(4)`, `attributes(5)`, `environ(5)`, `locale(5)`

NOTES

To change locale in a multithreaded application, `setlocale()` should be called prior to using any locale-sensitive routine. Using `setlocale()` to query the current locale is safe and can be used anywhere in a multithreaded application.

It is the user's responsibility to ensure that mixed locale categories are compatible. For example, setting `LC_CTYPE=C` and `LC_TIME=ja` (where `ja` indicates Japanese) will not work, because Japanese time cannot be represented in the "C" locale's ASCII codeset.

Internationalization functions by `setlocale()` are supported only when the dynamic linking version of `libc` has been linked with the application. If the static linking version of `libc` has been linked with the application, `setlocale()` can handle only C and POSIX locales.

| | |
|--------------------|--|
| NAME | setsockopt – set the socket options |
| SYNOPSIS | <pre>cc [flag ...] file ... -lxnet [library ...] #include <sys/socket.h></pre> <p>int setsockopt(int <i>socket</i>, int <i>level</i>, int <i>option_name</i>, const void*<i>option_value</i>, socklen_t <i>option_len</i>);</p> |
| DESCRIPTION | <p>The setsockopt() function sets the option specified by the <i>option_name</i> argument, at the protocol level specified by the <i>level</i> argument, to the value pointed to by the <i>option_value</i> argument for the socket associated with the file descriptor specified by the <i>socket</i> argument.</p> <p>The <i>level</i> argument specifies the protocol level at which the option resides. To set options at the socket level, specify the <i>level</i> argument as SOL_SOCKET. To set options at other levels, supply the appropriate protocol number for the protocol controlling the option. For example, to indicate that an option will be interpreted by the TCP (Transport Control Protocol), set <i>level</i> to the protocol number of TCP, as defined in the <code><netinet/in.h></code> header, or as determined by using getprotobyname(3XN).</p> <p>The <i>option_name</i> argument specifies a single option to set. The <i>option_name</i> argument and any specified options are passed uninterpreted to the appropriate protocol module for interpretations. The <code><sys/socket.h></code> header defines the socket level options. The options are as follows:</p> <p>SO_DEBUG Turns on recording of debugging information. This option enables or disables debugging in the underlying protocol modules. This option takes an <code>int</code> value. This is a boolean option.</p> <p>SO_BROADCAST Permits sending of broadcast messages, if this is supported by the protocol. This option takes an <code>int</code> value. This is a boolean option.</p> <p>SO_REUSEADDR Specifies that the rules used in validating addresses supplied to bind(3XN) should allow reuse of local addresses, if this is supported by the protocol. This option takes an <code>int</code> value. This is a boolean option.</p> <p>SO_KEEPAIVE Keeps connections active by enabling the periodic transmission of messages, if this is supported by the protocol. This option takes an <code>int</code> value.</p> <p>If the connected socket fails to respond to these messages, the connection is broken and processes</p> |

writing to that socket are notified with a SIGPIPE signal.

This is a boolean option.

SO_LINGER

Lingers on a `close(2)` if data is present. This option controls the action taken when unsent messages queue on a socket and `close(2)` is performed. If `SO_LINGER` is set, the system blocks the process during `close(2)` until it can transmit the data or until the time expires. If `SO_LINGER` is not specified, and `close(2)` is issued, the system handles the call in a way that allows the process to continue as quickly as possible. This option takes a `linger` structure, as defined in the `<sys/socket.h>` header, to specify the state of the option and linger interval.

SO_OOBINLINE

Leaves received out-of-band data (data marked urgent) in line. This option takes an `int` value. This is a boolean option.

SO_SNDBUF

Sets send buffer size. This option takes an `int` value.

SO_RCVBUF

Sets receive buffer size. This option takes an `int` value.

SO_DONTROUTE

Requests that outgoing messages bypass the standard routing facilities. The destination must be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. The effect, if any, of this option depends on what protocol is in use. This option takes an `int` value. This is a boolean option.

For boolean options, 0 indicates that the option is disabled and 1 indicates that the option is enabled.

Options at other protocol levels vary in format and name.

USAGE

The `setsockopt()` function provides an application program with the means to control socket behavior. An application program can use `setsockopt()` to allocate buffer space, control timeouts, or permit socket data broadcasts. The `<sys/socket.h>` header defines the socket-level options available to `setsockopt()`.

Options may exist at multiple protocol levels. The `SO_` options are always present at the uppermost socket level.

RETURN VALUES

Upon successful completion, **setsockopt()** returns 0. Otherwise, -1 is returned and `errno` is set to indicate the error.

ERRORS

The **setsockopt()** function will fail if:

- EBADF** The *socket* argument is not a valid file descriptor.
- EDOM** The send and receive timeout values are too big to fit into the timeout fields in the socket structure.
- EFAULT** The *option_value* parameter can not be accessed or written.
- EINVAL** The specified option is invalid at the specified socket level or the socket has been shut down.
- EISCONN** The socket is already connected, and a specified option can not be set while the socket is connected.
- ENOPROTOOPT** The option is not supported by the protocol.
- ENOTSOCK** The *socket* argument does not refer to a socket.
- The **setsockopt()** function may fail if:
- ENOMEM** There was insufficient memory available for the operation to complete.
- ENOBUFS** Insufficient resources are available in the system to complete the call.
- ENOSR** There were insufficient STREAMS resources available for the operation to complete.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

bind(3XN), **endprotoent(3XN)**, **getsockopt(3XN)**, **socket(3XN)**, **attributes(5)**

| | |
|----------------------|---|
| NAME | set_term – switch between terminals |
| SYNOPSIS | <pre>#include <curses.h> SCREEN *set_term(SCREEN *new);</pre> |
| PARAMETERS | <p><i>new</i> Is the new terminal to which the set_term() function will switch.</p> |
| DESCRIPTION | The set_term() function switches to the terminal specified by <i>new</i> and returns a screen reference to the previous terminal. Calls to subsequent X/Open Curses functions affect the new terminal. |
| RETURN VALUES | On success, the set_term() function returns a pointer to the previous screen. Otherwise, it returns a null pointer. |
| ERRORS | None. |

| | | | | | | | |
|-----------------------|--|-----------------------|----------------------------|---------------------|--------------------------------|----------------------|--|
| NAME | shm_open – open a shared memory object | | | | | | |
| SYNOPSIS | <pre>cc [<i>flag...</i>] <i>file...</i> -lrt [<i>library...</i>] #include <sys/mman.h> int shm_open(const char *<i>name</i>, int <i>oflag</i>, mode_t <i>mode</i>);</pre> | | | | | | |
| DESCRIPTION | <p>The shm_open() function establishes a connection between a shared memory object and a file descriptor. It creates an open file description that refers to the shared memory object and a file descriptor that refers to that open file description. The file descriptor is used by other functions to refer to that shared memory object. The <i>name</i> argument points to a string naming a shared memory object. It is unspecified whether the name appears in the file system and is visible to other functions that take pathnames as arguments. The <i>name</i> argument conforms to the construction rules for a pathname. The first character of <i>name</i> must be a slash (/) character and the remaining characters of <i>name</i> cannot include any slash characters. For maximum portability, <i>name</i> should include no more than 14 characters, but this limit is not enforced.</p> <p>If successful, shm_open() returns a file descriptor for the shared memory object that is the lowest numbered file descriptor not currently open for that process. The open file description is new, and therefore the file descriptor does not share it with any other processes. It is unspecified whether the file offset is set. The <code>FD_CLOEXEC</code> file descriptor flag associated with the new file descriptor is set.</p> <p>The file status flags and file access modes of the open file description are according to the value of <i>oflag</i>. The <i>oflag</i> argument is the bitwise inclusive OR of the following flags defined in the header <code><fcntl.h></code>. Applications specify exactly one of the first two values (access modes) below in the value of <i>oflag</i>:</p> <table border="0"> <tr> <td style="padding-right: 20px;"><code>O_RDONLY</code></td> <td>Open for read access only.</td> </tr> <tr> <td><code>O_RDWR</code></td> <td>Open for read or write access.</td> </tr> </table> <p>Any combination of the remaining flags may be specified in the value of <i>oflag</i>:</p> <table border="0"> <tr> <td style="padding-right: 20px;"><code>O_CREAT</code></td> <td>If the shared memory object exists, this flag has no effect, except as noted under <code>O_EXCL</code> below. Otherwise the shared memory object is created; the user ID of the shared memory object will be set to the effective user ID of the process; the group ID of the shared memory object will be set to a system default group ID or to the effective group ID of the process. The permission bits of the shared memory object will be set to the value of the <i>mode</i> argument except those set in the file mode creation mask of the process. When bits in <i>mode</i> other than the file permission bits are set, the effect is unspecified. The <i>mode</i> argument does not affect whether the shared memory object is opened for reading, for writing, or for both. The shared memory object has a size of zero.</td> </tr> </table> | <code>O_RDONLY</code> | Open for read access only. | <code>O_RDWR</code> | Open for read or write access. | <code>O_CREAT</code> | If the shared memory object exists, this flag has no effect, except as noted under <code>O_EXCL</code> below. Otherwise the shared memory object is created; the user ID of the shared memory object will be set to the effective user ID of the process; the group ID of the shared memory object will be set to a system default group ID or to the effective group ID of the process. The permission bits of the shared memory object will be set to the value of the <i>mode</i> argument except those set in the file mode creation mask of the process. When bits in <i>mode</i> other than the file permission bits are set, the effect is unspecified. The <i>mode</i> argument does not affect whether the shared memory object is opened for reading, for writing, or for both. The shared memory object has a size of zero. |
| <code>O_RDONLY</code> | Open for read access only. | | | | | | |
| <code>O_RDWR</code> | Open for read or write access. | | | | | | |
| <code>O_CREAT</code> | If the shared memory object exists, this flag has no effect, except as noted under <code>O_EXCL</code> below. Otherwise the shared memory object is created; the user ID of the shared memory object will be set to the effective user ID of the process; the group ID of the shared memory object will be set to a system default group ID or to the effective group ID of the process. The permission bits of the shared memory object will be set to the value of the <i>mode</i> argument except those set in the file mode creation mask of the process. When bits in <i>mode</i> other than the file permission bits are set, the effect is unspecified. The <i>mode</i> argument does not affect whether the shared memory object is opened for reading, for writing, or for both. The shared memory object has a size of zero. | | | | | | |

| | |
|----------------|--|
| O_EXCL | If O_EXCL and O_CREAT are set, shm_open() fails if the shared memory object exists. The check for the existence of the shared memory object and the creation of the object if it does not exist is atomic with respect to other processes executing shm_open() naming the same shared memory object with O_EXCL and O_CREAT set. If O_EXCL is set and O_CREAT is not set, the result is undefined. |
| O_TRUNC | If the shared memory object exists, and it is successfully opened O_RDWR , the object will be truncated to zero length and the mode and owner will be unchanged by this function call. The result of using O_TRUNC with O_RDONLY is undefined. |

When a shared memory object is created, the state of the shared memory object, including all data associated with the shared memory object, persists until the shared memory object is unlinked and all other references are gone. It is unspecified whether the name and shared memory object state remain valid after a system reboot.

RETURN VALUES

Upon successful completion, the **shm_open()** function returns a non-negative integer representing the lowest numbered unused file descriptor. Otherwise, it returns `-1` and sets `errno` to indicate the error condition.

ERRORS

The **shm_open()** function will fail if:

| | |
|---------------|--|
| EACCES | The shared memory object exists and the permissions specified by <i>oflag</i> are denied, or the shared memory object does not exist and permission to create the shared memory object is denied, or O_TRUNC is specified and write permission is denied. |
| EEXIST | O_CREAT and O_EXCL are set and the named shared memory object already exists. |
| EINTR | The shm_open() operation was interrupted by a signal. |
| EINVAL | The shm_open() operation is not supported for the given name. |
| EMFILE | Too many file descriptors are currently in use by this process. |

- ENAMETOOLONG** The length of the *name* string exceeds `PATH_MAX`, or a pathname component is longer than `NAME_MAX` while `_POSIX_NO_TRUNC` is in effect.
- ENFILE** Too many shared memory objects are currently open in the system.
- ENOENT** `O_CREAT` is not set and the named shared memory object does not exist.
- ENOSPC** There is insufficient space for the creation of the new shared memory object.
- ENOSYS** The `shm_open()` function is not supported by the system.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`close(2)`, `dup(2)`, `exec(2)`, `fcntl(2)`, `mmap(2)`, `umask(2)`, `shm_unlink(3R)`, `sysconf(3C)`, `attributes(5)`, `fcntl(5)`

NOTES

Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned `-1` and set `errno` to `ENOSYS`.

| NAME | shm_unlink – remove a shared memory object | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <sys/mman.h> int shm_unlink(const char *name);</pre> | | | | |
| DESCRIPTION | The shm_unlink() function removes the name of the shared memory object named by the string pointed to by <i>name</i> . If one or more references to the shared memory object exists when the object is unlinked, the name is removed before shm_unlink() returns, but the removal of the memory object contents will be postponed until all open and mapped references to the shared memory object have been removed. | | | | |
| RETURN VALUES | Upon successful completion, shm_unlink() returns 0. Otherwise it returns -1 and sets <i>errno</i> to indicate the error condition, and the named shared memory object is not affected by this function call. | | | | |
| ERRORS | <p>The shm_unlink() function will fail if:</p> <p>EACCES Permission is denied to unlink the named shared memory object.</p> <p>ENAMETOOLONG The length of the <i>name</i> string exceeds <i>PATH_MAX</i>, or a pathname component is longer than <i>NAME_MAX</i> while <i>_POSIX_NO_TRUNC</i> is in effect.</p> <p>ENOENT The named shared memory object does not exist.</p> <p>ENOSYS The shm_unlink() function is not supported by the system.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | close(2) , mmap(2) , mlock(3C) , shm_open(3R) , attributes(5) | | | | |
| NOTES | Solaris 2.6 was the first release to support the Asynchronous Input and Output option. Prior to this release, this function always returned -1 and set <i>errno</i> to <i>ENOSYS</i> . | | | | |

NAME shutdown – shut down part of a full-duplex connection

SYNOPSIS `cc [flag ...] file ... -lsocket -lnsl [library ...]`

```
int shutdown(int s, int how);
```

DESCRIPTION The **shutdown()** call shuts down all or part of a full-duplex connection on the socket associated with *s*. If *how* is 0, then further receives will be disallowed. If *how* is 1, then further sends will be disallowed. If *how* is 2, then further sends and receives will be disallowed.

RETURN VALUES A 0 is returned if the call succeeds, -1 if it fails.

ERRORS The call succeeds unless:

EBADF *s* is not a valid file descriptor.

ENOMEM There was insufficient user memory available for the operation to complete.

ENOSR There were insufficient STREAMS resources available for the operation to complete.

ENOTCONN The specified socket is not connected.

ENOTSOCK *s* is not a socket.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO **connect(3N)**, **socket(3N)**, **attributes(5)**, **socket(5)**

NOTES The *how* values should be defined constants.

| NAME | shutdown – shut down socket send and receive operations | | | | | | | | | | | | |
|----------------------|---|----------------|--|----------------|-------------------------------------|------------------|---|-----------------|--|----------------|---|--------------|--|
| SYNOPSIS | <pre>cc [flag ...] file ... -lxnet [library ...] #include <sys/socket.h> int shutdown(int socket, int how);</pre> | | | | | | | | | | | | |
| DESCRIPTION | <p>socket Specifies the file descriptor of the socket.</p> <p>how Specifies the type of shutdown. The values are as follows:</p> <table border="0"> <tr> <td style="padding-left: 2em;">SHUT_RD</td> <td>Disables further receive operations.</td> </tr> <tr> <td style="padding-left: 2em;">SHUT_WR</td> <td>Disables further send operations.</td> </tr> <tr> <td style="padding-left: 2em;">SHUT_RDWR</td> <td>Disables further send and receive operations.</td> </tr> </table> <p>The shutdown() function disables subsequent send and/or receive operations on a socket, depending on the value of the <i>how</i> argument.</p> | SHUT_RD | Disables further receive operations. | SHUT_WR | Disables further send operations. | SHUT_RDWR | Disables further send and receive operations. | | | | | | |
| SHUT_RD | Disables further receive operations. | | | | | | | | | | | | |
| SHUT_WR | Disables further send operations. | | | | | | | | | | | | |
| SHUT_RDWR | Disables further send and receive operations. | | | | | | | | | | | | |
| RETURN VALUES | Upon successful completion, shutdown() returns 0. Otherwise, -1 is returned and <i>errno</i> is set to indicate the error. | | | | | | | | | | | | |
| ERRORS | <p>The shutdown() function will fail if:</p> <table border="0"> <tr> <td style="padding-left: 2em;">EBADF</td> <td>The <i>socket</i> argument is not a valid file descriptor.</td> </tr> <tr> <td style="padding-left: 2em;">EINVAL</td> <td>The <i>how</i> argument is invalid.</td> </tr> <tr> <td style="padding-left: 2em;">ENOTCONN</td> <td>The socket is not connected.</td> </tr> <tr> <td style="padding-left: 2em;">ENOTSOCK</td> <td>The <i>socket</i> argument does not refer to a socket.</td> </tr> </table> <p>The shutdown() function may fail if:</p> <table border="0"> <tr> <td style="padding-left: 2em;">ENOBUFS</td> <td>Insufficient resources were available in the system to perform the operation.</td> </tr> <tr> <td style="padding-left: 2em;">ENOSR</td> <td>There were insufficient STREAMS resources available for the operation to complete.</td> </tr> </table> | EBADF | The <i>socket</i> argument is not a valid file descriptor. | EINVAL | The <i>how</i> argument is invalid. | ENOTCONN | The socket is not connected. | ENOTSOCK | The <i>socket</i> argument does not refer to a socket. | ENOBUFS | Insufficient resources were available in the system to perform the operation. | ENOSR | There were insufficient STREAMS resources available for the operation to complete. |
| EBADF | The <i>socket</i> argument is not a valid file descriptor. | | | | | | | | | | | | |
| EINVAL | The <i>how</i> argument is invalid. | | | | | | | | | | | | |
| ENOTCONN | The socket is not connected. | | | | | | | | | | | | |
| ENOTSOCK | The <i>socket</i> argument does not refer to a socket. | | | | | | | | | | | | |
| ENOBUFS | Insufficient resources were available in the system to perform the operation. | | | | | | | | | | | | |
| ENOSR | There were insufficient STREAMS resources available for the operation to complete. | | | | | | | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | | | | | | | | | |
| | <table border="1"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe | | | | | | | | |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | | | | | | | |
| MT-Level | MT-Safe | | | | | | | | | | | | |

SEE ALSO

`getsockopt(3XN)`, `recv(3XN)`, `recvfrom(3XN)`, `recvmsg(3XN)`,
`select(3C)`, `send(3XN)`, `sendto(3XN)`, `setsockopt(3XN)`, `socket(3XN)`,
`attributes(5)`

| | |
|--------------------|---|
| NAME | sigblock, sigmask, sigpause, sigsetmask – block signals |
| SYNOPSIS | <pre> /usr/ucb/cc [flag ...] file ... #include <signal.h> int sigblock(mask); int mask ; int sigmask(signum); int signum ; int sigpause(int mask); int mask ; int sigsetmask(mask); int mask ; </pre> |
| DESCRIPTION | <p>sigblock, sigmask, sigpause, sigsetmask – block signals</p> <p>sigblock() adds the signals specified in <i>mask</i> to the set of signals currently being blocked from delivery. Signals are blocked if the appropriate bit in <i>mask</i> is a 1; the macro <code>sigmask</code> is provided to construct the mask for a given <i>signum</i>. sigblock() returns the previous mask. The previous mask may be restored using sigsetmask().</p> <p>sigpause() assigns <i>mask</i> to the set of masked signals and then waits for a signal to arrive; on return the set of masked signals is restored. <i>mask</i> is usually 0 to indicate that no signals are now to be blocked. sigpause() always terminates by being interrupted, returning -1 and setting <code>errno</code> to <code>EINTR</code>.</p> <p>sigsetmask() sets the current signal mask (those signals that are blocked from delivery). Signals are blocked if the corresponding bit in <i>mask</i> is a 1; the macro <code>sigmask</code> is provided to construct the mask for a given <i>signum</i>.</p> |

In normal usage, a signal is blocked using **sigblock()** . To begin a critical section, variables modified on the occurrence of the signal are examined to determine that there is no work to be done, and the process pauses awaiting work by using **sigpause()** with the mask returned by **sigblock()** .

It is not possible to block SIGKILL , SIGSTOP , or SIGCONT , this restriction is silently imposed by the system.

RETURN VALUES

sigblock() and **sigsetmask()** return the previous set of masked signals. **sigpause()** returns -1 and sets *errno* to EINTR.

SEE ALSO

kill(2) , **sigaction(2)** , **signal(3B)** , **sigvec(3B)**

NOTES

Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

| | |
|--------------------|---|
| NAME | sigfpe – signal handling for specific SIGFPE codes |
| SYNOPSIS | <pre>#include <floatingpoint.h> #include <siginfo.h></pre> <p>sigfpe_handler_type sigfpe(sigfpe_code_type <i>code</i>, sigfpe_handler_type <i>hdl</i>);</p> |
| DESCRIPTION | <p>This function allows signal handling to be specified for particular SIGFPE codes. A call to sigfpe() defines a new handler <i>hdl</i> for a particular SIGFPE <i>code</i> and returns the old handler as the value of the function sigfpe(). Normally handlers are specified as pointers to functions; the special cases SIGFPE_IGNORE, SIGFPE_ABORT, and SIGFPE_DEFAULT allow ignoring, dumping core using abort(3C), or default handling respectively. Default handling is to dump core using abort(3C).</p> <p><i>code</i> is usually one of the five IEEE 754-related SIGFPE codes:</p> <pre>FPE_FLTRES fp_inexact - floating-point inexact result FPE_FLTDIV fp_division - floating-point division by zero FPE_FLTUND fp_underflow - floating-point underflow FPE_FLTOVF fp_overflow - floating-point overflow FPE_FLTINV fp_invalid - floating-point invalid operation</pre> <p>Three steps are required to intercept an IEEE 754-related SIGFPE code with sigfpe():</p> <ol style="list-style-type: none"> 1. Set up a handler with sigfpe(). 2. Enable the relevant IEEE 754 trapping capability in the hardware, perhaps by using assembly-language instructions. 3. Perform a floating-point operation that generates the intended IEEE 754 exception. <p>sigfpe() never changes floating-point hardware mode bits affecting IEEE 754 trapping. No IEEE 754-related SIGFPE signals will be generated unless those hardware mode bits are enabled.</p> <p>SIGFPE signals can be handled using sigfpe(), sigaction(2) or signal(3C). In a particular program, to avoid confusion, use only one of these interfaces to handle SIGFPE signals.</p> |
| EXAMPLES | <p>EXAMPLE 1 Example Of A User-Specified Signal Handler</p> <p>A user-specified signal handler might look like this:</p> |

```

#include <floatingpoint.h>
#include <siginfo.h>
#include <ucontext.h>
/*
 * The sample_handler prints out a message then commits suicide.
 */
void
sample_handler(int sig, siginfo_t *sip, ucontext_t *uap) {
    char *label;
    switch (sip->si_code) {
    case FPE_FLTINV: label = "invalid operand"; break;
    case FPE_FLTRES: label = "inexact"; break;
    case FPE_FLTDIV: label = "division-by-zero"; break;
    case FPE_FLTUND: label = "underflow"; break;
    case FPE_FLTOVF: label = "overflow"; break;
    default: label = "???"; break;
    }
    fprintf(stderr, "FP exception %s (0x%x) occurred at address %p.\n",
            label, sip->si_code, (void *) sip->si_addr);
    abort();
}

```

and it might be set up like this:

```

#include <floatingpoint.h>
#include <siginfo.h>
#include <ucontext.h>
extern void sample_handler(int, siginfo_t *, ucontext_t *);
main(void) {
    sigfpe_handler_type hdl, old_handler1, old_handler2;
    /*
     * save current fp_overflow and fp_invalid handlers; set the new
     * fp_overflow handler to sample_handler() and set the new
     * fp_invalid handler to SIGFPE_ABORT (abort on invalid)
     */
    hdl = (sigfpe_handler_type) sample_handler;
    old_handler1 = sigfpe(FPE_FLTOVF, hdl);
    old_handler2 = sigfpe(FPE_FLTINV, SIGFPE_ABORT);
    ...
    /*
     * restore old fp_overflow and fp_invalid handlers
     */
    sigfpe(FPE_FLTOVF, old_handler1);
    sigfpe(FPE_FLTINV, old_handler2);
}

```

FILES

/usr/include/floatingpoint.h

/usr/include/siginfo.h

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

sigaction(2), **abort(3C)**, **signal(3C)**, **attributes(5)**,
floatingpoint(5)

DIAGNOSTICS

sigfpe() returns BADSIG if *code* is not zero or a defined SIGFPE code.

| | |
|----------------------|---|
| NAME | siginterrupt – allow signals to interrupt functions |
| SYNOPSIS | <pre>/usr/ucb/cc [<i>flag ...</i>] <i>file ...</i> int siginterrupt(<i>sig, flag</i>); int <i>sig, flag</i>;</pre> |
| DESCRIPTION | <p>siginterrupt() is used to change the function restart behavior when a function is interrupted by the specified signal. If the flag is false (0), then functions will be restarted if they are interrupted by the specified signal and no data has been transferred yet. System call restart is the default behavior when the signal(3C) routine is used.</p> <p>If the flag is true, (1), then restarting of functions is disabled. If a function is interrupted by the specified signal and no data has been transferred, the function will return -1 with errno set to EINTR. Interrupted functions that have started transferring data will return the amount of data actually transferred.</p> <p>Issuing a siginterrupt() call during the execution of a signal handler will cause the new action to take place on the next signal to be caught.</p> |
| NOTES | <p>Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-threaded applications is unsupported.</p> <p>This library routine uses an extension of the sigvec(3B) function that is not available in 4.2 BSD, hence it should not be used if backward compatibility is needed.</p> |
| RETURN VALUES | A 0 value indicates that the call succeeded. A -1 value indicates that the call failed and errno is set to indicate the error. |
| ERRORS | siginterrupt() may return the following error: EINVAL <i>sig</i> is not a valid signal. |
| SEE ALSO | sigblock(3B) , sigvec(3B) , signal(3C) |

| | |
|--------------------|--|
| NAME | signal – simplified software signal facilities |
| SYNOPSIS | <pre> /usr/ucb/cc [flag ...] file ... #include <signal.h> void (*signal(sig, func))(); int sig; void (*func)(); </pre> |
| DESCRIPTION | <p>signal() is a simplified interface to the more general sigvec(3B) facility. Programs that use signal() in preference to sigvec() are more likely to be portable to all systems.</p> <p>A signal is generated by some abnormal event, initiated by a user at a terminal (quit, interrupt, stop), by a program error (bus error, etc.), by request of another program (kill), or when a process is stopped because it wishes to access its control terminal while in the background (see termio(7I)). Signals are optionally generated when a process resumes after being stopped, when the status of child processes changes, or when input is ready at the control terminal. Most signals cause termination of the receiving process if no action is taken; some signals instead cause the process receiving them to be stopped, or are simply discarded if the process has not requested otherwise. Except for the SIGKILL and SIGSTOP signals, the signal() call allows signals either to be ignored or to interrupt to a specified location. See sigvec(3B) for a complete list of the signals.</p> <p>If <i>func</i> is SIG_DFL, the default action for signal <i>sig</i> is reinstated; this default is termination (with a core image for starred signals) except for signals marked with • or a dagger.. Signals marked with • are discarded if the action is SIG_DFL; signals marked with a dagger cause the process to stop. If <i>func</i> is SIG_IGN the signal is subsequently ignored and pending instances of the signal are discarded. Otherwise, when the signal occurs further occurrences of the signal are automatically blocked and <i>func</i> is called.</p> <p>A return from the function unblocks the handled signal and continues the process at the point it was interrupted.</p> <p>If a caught signal occurs during certain functions, terminating the call prematurely, the call is automatically restarted. In particular this can occur during a read(2) or write(2) on a slow device (such as a terminal; but not a file) and during a wait(2).</p> <p>The value of signal() is the previous (or initial) value of <i>func</i> for the particular signal.</p> <p>After a fork(2) or vfork(2) the child inherits all signals. An exec(2) resets all caught signals to the default action; ignored signals remain ignored.</p> |

RETURN VALUES The previous action is returned on a successful call. Otherwise, `-1` is returned and `errno` is set to indicate the error.

ERRORS **signal()** will fail and no action will take place if the following occurs:
EINVAL `sig` is not a valid signal number, or is `SIGKILL` or `SIGSTOP`.

SEE ALSO `kill(1)`, `exec(2)`, `fcntl(2)`, `fork(2)`, `getitimer(2)`, `getrlimit(2)`, `kill(2)`, `ptrace(2)`, `read(2)`, `sigaction(2)`, `wait(2)`, `write(2)`, `abort(3C)`, `setjmp(3B)`, `sigblock(3B)`, `sigstack(3B)`, `sigvec(3B)`, `wait(3B)`, `setjmp(3C)`, `signal(3C)`, `signal(5)`, `termio(7I)`

NOTES Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-threaded applications is unsupported.

The handler routine, *func*, can be declared:

```
void handler( signum) int signum;
```

Here *signum* is the signal number. See `sigvec(3B)` for more details.

| | |
|----------------------|---|
| NAME | signal, sigset, sighold, sigrelse, sigignore, sigpause – simplified signal management for application processes |
| SYNOPSIS | <pre>#include <signal.h> void (*signal (int sig, void (* disp)(int)))(int); void (* sigset(int sig, void (* disp)(int)))(int); int sighold(int sig); int sigrelse(int sig); int sigignore(int sig); int sigpause(int sig);</pre> |
| DESCRIPTION | <p>These functions provide simplified signal management for application processes. See signal(5) for an explanation of general signal concepts.</p> <p>The signal() and sigset() functions modify signal dispositions. The <i>sig</i> argument specifies the signal, which may be any signal except SIGKILL and SIGSTOP. The <i>disp</i> argument specifies the signal's disposition, which may be SIG_DFL, SIG_IGN, or the address of a signal handler. If signal() is used, <i>disp</i> is the address of a signal handler, and <i>sig</i> is not SIGILL, SIGTRAP, or SIGPWR, the system first sets the signal's disposition to SIG_DFL before executing the signal handler. If sigset() is used and <i>disp</i> is the address of a signal handler, the system adds <i>sig</i> to the calling process's signal mask before executing the signal handler; when the signal handler returns, the system restores the calling process's signal mask to its state prior to the delivery of the signal. In addition, if sigset() is used and <i>disp</i> is equal to SIG_HOLD, <i>sig</i> is added to the calling process's signal mask and the signal's disposition remains unchanged.</p> <p>The sighold() function adds <i>sig</i> to the calling process's signal mask.</p> <p>The sigrelse() function removes <i>sig</i> from the calling process's signal mask.</p> <p>The sigignore() function sets the disposition of <i>sig</i> to SIG_IGN.</p> <p>The sigpause() function removes <i>sig</i> from the calling process's signal mask and suspends the calling process until a signal is received.</p> |
| RETURN VALUES | <p>Upon successful completion, signal() returns the signal's previous disposition. Otherwise, it returns SIG_ERR and sets <i>errno</i> to indicate the error.</p> <p>Upon successful completion, sigset() returns SIG_HOLD if the signal had been blocked or the signal's previous disposition if it had not been blocked. Otherwise, it returns SIG_ERR and sets <i>errno</i> to indicate the error.</p> |

Upon successful completion, **sighold()** , **sigrelse()** , **sigignore()** , and **sigpause()** , return 0 . Otherwise, they return -1 and set `errno` to indicate the error.

ERRORS

These functions fail if:

- EINTR** A signal was caught during the execution **sigpause()** .
- EINVAL** The value of the *sig* argument is not a valid signal or is equal to SIGKILL or SIGSTOP .

USAGE

The **sighold()** function used in conjunction with **sigrelse()** or **sigpause()** may be used to establish critical regions of code that require the delivery of a signal to be temporarily deferred.

If **signal()** or **sigset()** is used to set SIGCHLD 's disposition to a signal handler, SIGCHLD will not be sent when the calling process's children are stopped or continued.

If any of the above functions are used to set SIGCHLD 's disposition to SIG_IGN, the calling process's child processes will not create zombie processes when they terminate (see **exit(2)**). If the calling process subsequently waits for its children, it blocks until all of its children terminate; it then returns -1 with `errno` set to ECHILD (see **wait(2)** and **waitid(2)**).

The system guarantees that if more than one instance of the same signal is generated to a process, at least one signal will be received. It does not guarantee the reception of every generated signal.

SEE ALSO

exit(2) , **kill(2)** , **pause(2)** , **sigaction(2)** , **sigsend(2)** , **wait(2)** , **waitid(2)** , **signal(5)**

NAME | significand – significand function

SYNOPSIS | `cc [flag ...] file ... -lm [library ...]`
 | `#include <math.h>`
 | `double significand(double x);`

DESCRIPTION | The **significand()** function, along with the **logb(3M)** and **scalb(3M)** functions, allows users to verify compliance to ANSI/IEEE Std 754-1985 by running certain test vectors distributed by the University of California.

| If x equals $sig * 2^{**n}$ with $1 < sig < 2$, then **significand(*x*)** returns *sig* for exercising the fraction-part(F) test vector. **significand(*x*)** is not defined when x is either 0, ±Inf or NaN.

RETURN VALUES | For exceptional cases, **matherr(3M)** tabulates the values to be returned as dictated by various Standards.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **logb(3M)**, **matherr(3M)**, **scalb(3M)**, **attributes(5)**

| | |
|----------------------|--|
| NAME | sigqueue – queue a signal to a process |
| SYNOPSIS | <pre>cc [<i>flag...</i>] <i>file...</i> -lrt [<i>library...</i>] #include <sys/types.h> #include <signal.h> int sigqueue(pid_t <i>pid</i>, int <i>signo</i>, const union signal <i>value</i>);</pre> |
| DESCRIPTION | <p>The sigqueue() function causes the signal specified by <i>signo</i> to be sent with the value specified by <i>value</i> to the process specified by <i>pid</i>. If <i>signo</i> is 0 (the null signal), error checking is performed but no signal is actually sent. The null signal can be used to check the validity of <i>pid</i>.</p> <p>The conditions required for a process to have permission to queue a signal to another process are the same as for the kill(2) function.</p> <p>The sigqueue() function returns immediately. If SA_SIGINFO is set for <i>signo</i> and if the resources were available to queue the signal, the signal is queued and sent to the receiving process. If SA_SIGINFO is not set for <i>signo</i>, then <i>signo</i> is sent at least once to the receiving process; it is unspecified whether <i>value</i> will be sent to the receiving process as a result of this call.</p> <p>If the value of <i>pid</i> causes <i>signo</i> to be generated for the sending process, and if <i>signo</i> is not blocked for the calling thread and if no other thread has <i>signo</i> unblocked or is waiting in a sigwait(2) function for <i>signo</i>, either <i>signo</i> or at least the pending, unblocked signal will be delivered to the calling thread before the sigqueue() function returns. Should any of multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected for delivery, it will be the lowest numbered one. The selection order between realtime and non-realtime signals, or between multiple pending non-realtime signals, is unspecified.</p> |
| RETURN VALUES | Upon successful completion, the specified signal will have been queued, and the sigqueue() function returns 0. Otherwise, the function returns -1 and sets <code>errno</code> to indicate the error. |
| ERRORS | <p>The sigqueue() function will fail if:</p> <p>EAGAIN No resources are available to queue the signal. The process has already queued SIGQUEUE_MAX signals that are still pending at the receiver(s), or a system wide resource limit has been exceeded.</p> <p>EINVAL The value of <i>signo</i> is an invalid or unsupported signal number.</p> <p>ENOSYS The sigqueue() function is not supported by the system.</p> <p>EPERM The process does not have the appropriate privilege to send the signal to the receiving process.</p> |

ESRCH The process *pid* does not exist.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------|
| MT-Level | Async-Signal-Safe |

SEE ALSO

kill(2), **sigwaitinfo(3R)**, **attributes(5)**, **siginfo(5)**, **signal(5)**

| | |
|----------------------|--|
| NAME | sigsetops, sigemptyset, sigfillset, sigaddset, sigdelset, sigismember – manipulate sets of signals |
| SYNOPSIS | <pre>#include <signal.h> int sigemptyset(sigset_t * set); int sigfillset(sigset_t * set); int sigaddset(sigset_t * set, int signo); int sigdelset(sigset_t * set, int signo); int sigismember(sigset_t * set, int signo);</pre> |
| DESCRIPTION | <p>These functions manipulate <code>sigset_t</code> data types, representing the set of signals supported by the implementation.</p> <p>The sigemptyset() function initializes the set pointed to by <i>set</i> to exclude all signals defined by the system.</p> <p>The sigfillset() function initializes the set pointed to by <i>set</i> to include all signals defined by the system.</p> <p>The sigaddset() function adds the individual signal specified by the value of <i>signo</i> to the set pointed to by <i>set</i> .</p> <p>The sigdelset() function deletes the individual signal specified by the value of <i>signo</i> from the set pointed to by <i>set</i> .</p> <p>The sigismember() function checks whether the signal specified by the value of <i>signo</i> is a member of the set pointed to by <i>set</i> .</p> <p>Any object of type <i>sigset_t</i> must be initialized by applying either sigemptyset() or sigfillset() before applying any other operation.</p> |
| RETURN VALUES | <p>Upon successful completion, the sigismember() function returns 1 if the specified signal is a member of the specified set, or 0 if it is not.</p> <p>Upon successful completion, the other functions return 0 . Otherwise -1 is returned and <code>errno</code> is set to indicate the error.</p> |
| ERRORS | <p>The sigaddset() , sigdelset() , and sigismember() functions will fail if:</p> <p>EINVAL The value of the <i>signo</i> argument is not a valid signal number.</p> <p>The sigfillset() function will fail if:</p> <p>EFAULT The <i>set</i> argument specifies an invalid address.</p> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`sigaction(2)`, `sigpending(2)`, `sigprocmask(2)`, `sigsuspend(2)`,
`attributes(5)`, `signal(5)`

| | |
|----------------------|--|
| NAME | sigstack – set and/or get signal stack context |
| SYNOPSIS | <pre> /usr/ucb/cc [flag ...] file ... #include <signal.h> int sigstack(nss, oss); struct sigstack *nss, *oss; </pre> |
| DESCRIPTION | <p>The sigstack() function allows users to define an alternate stack, called the “signal stack“, on which signals are to be processed. When a signal’s action indicates its handler should execute on the signal stack (specified with a sigvec(3B) call), the system checks to see if the process is currently executing on that stack. If the process is not currently executing on the signal stack, the system arranges a switch to the signal stack for the duration of the signal handler’s execution.</p> <p>A signal stack is specified by a sigstack() structure, which includes the following members:</p> <pre> char *ss_sp; /* signal stack pointer */ int ss_onstack; /* current status */ </pre> <p>The <code>ss_sp</code> member is the initial value to be assigned to the stack pointer when the system switches the process to the signal stack. Note that, on machines where the stack grows downwards in memory, this is <i>not</i> the address of the beginning of the signal stack area. The <code>ss_onstack</code> member is zero or non-zero depending on whether the process is currently executing on the signal stack or not.</p> <p>If <code>nss</code> is not a null pointer, sigstack() sets the signal stack state to the value in the sigstack() structure pointed to by <code>nss</code>. If <code>nss</code> is a null pointer, the signal stack state will be unchanged. If <code>oss</code> is not a null pointer, the current signal stack state is stored in the sigstack() structure pointed to by <code>oss</code>.</p> |
| RETURN VALUES | Upon successful completion, 0 is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error. |
| ERRORS | <p>The sigstack() function will fail and the signal stack context will remain unchanged if one of the following occurs.</p> <p>EFAULT Either <code>nss</code> or <code>oss</code> points to memory that is not a valid part of the process address space.</p> |

SEE ALSO | `sigaltstack(2)`, `sigvec(3B)`, `signal(3C)`

WARNINGS | Signal stacks are not “grown” automatically, as is done for the normal stack. If the stack overflows unpredictable results may occur.

NOTES | Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-threaded applications is unsupported.

| | |
|----------------------|---|
| NAME | sigstack – set and/or get alternate signal stack context |
| SYNOPSIS | <pre>#include <signal.h> int sigstack(struct sigstack *ss, struct sigstack *oss);</pre> |
| DESCRIPTION | <p>The sigstack() function allows the calling process to indicate to the system an area of its address space to be used for processing signals received by the process.</p> <p>If the <i>ss</i> argument is not a null pointer, it must point to a <code>sigstack</code> structure. The length of the application-supplied stack must be at least <code>SIGSTKSZ</code> bytes. If the alternate signal stack overflows, the resulting behavior is undefined. (See <code>USAGE</code> below.)</p> <ul style="list-style-type: none"> ■ The value of the <code>ss_onstack</code> member indicates whether the process wants the system to use an alternate signal stack when delivering signals. ■ The value of the <code>ss_sp</code> member indicates the desired location of the alternate signal stack area in the process' address space. ■ If the <i>ss</i> argument is a null pointer, the current alternate signal stack context is not changed. <p>If the <i>oss</i> argument is not a null pointer, it points to a <code>sigstack</code> structure in which the current alternate signal stack context is placed. The value stored in the <code>ss_onstack</code> member of <i>oss</i> will be non-zero if the process is currently executing on the alternate signal stack. If the <i>oss</i> argument is a null pointer, the current alternate signal stack context is not returned.</p> <p>When a signal's action indicates its handler should execute on the alternate signal stack (specified by calling <code>sigaction(2)</code>), sigstack() checks to see if the process is currently executing on that stack. If the process is not currently executing on the alternate signal stack, the system arranges a switch to the alternate signal stack for the duration of the signal handler's execution.</p> <p>After a successful call to one of the <code>exec</code> functions, there are no alternate signal stacks in the new process image.</p> |
| RETURN VALUES | Upon successful completion, sigstack() returns 0. Otherwise, it returns -1 and sets <code>errno</code> to indicate the error. |
| ERRORS | <p>The sigstack() function will fail if:</p> <p>EPERM An attempt was made to modify an active stack.</p> |
| USAGE | A portable application, when being written or rewritten, should use <code>sigaltstack(2)</code> instead of sigstack() . |

The direction of stack growth is not indicated in the historical definition of `struct sigstack`. The only way to portably establish a stack pointer is for the application to determine stack growth direction, or to allocate a block of storage and set the stack pointer to the middle. **sigstack()** may assume that the size of the signal stack is `SIGSTKSZ` as found in `<signal.h>`. An application that would like to specify a signal stack size other than `SIGSTKSZ` should use **sigaltstack(2)**.

Applications should not use **longjmp(3C)** to leave a signal handler that is running on a stack established with **sigstack()**. Doing so may disable future use of the signal stack. For abnormal exit from a signal handler, **siglongjmp(3C)**, **setcontext(2)**, or **swapcontext(3C)** may be used. These functions fully support switching from one stack to another.

The **sigstack()** function requires the application to have knowledge of the underlying system's stack architecture. For this reason, **sigaltstack(2)** is recommended over this function.

SEE ALSO

fork(2), **_longjmp(3C)**, **longjmp(3C)**, **setjmp(3C)**, **sigaltstack(2)**, **siglongjmp(3C)**, **sigsetjmp(3C)**

| | |
|--------------------|---|
| NAME | sigvec – software signal facilities |
| SYNOPSIS | <pre>/usr/ucb/cc[flag ...] file... #include <signal.h> int sigvec(ssig, *nvec, *ovec); int sig; struct sigvec *nvec struct sigvec *ovec struct sigvec *nvec, *ovec;</pre> |
| DESCRIPTION | <p>The system defines a set of signals that may be delivered to a process. Signal delivery resembles the occurrence of a hardware interrupt: the signal is blocked from further occurrence, the current process context is saved, and a new one is built. A process may specify a <i>handler</i> to which a signal is delivered, or specify that a signal is to be <i>blocked</i> or <i>ignored</i>. A process may also specify that a default action is to be taken by the system when a signal occurs. Normally, signal handlers execute on the current stack of the process. This may be changed, on a per-handler basis, so that signals are taken on a special <i>signal stack</i>.</p> <p>All signals have the same <i>priority</i>. Signal routines execute with the signal that caused their invocation to be <i>blocked</i>, but other signals may yet occur. A global <i>signal mask</i> defines the set of signals currently blocked from delivery to a process. The signal mask for a process is initialized from that of its parent (normally 0). It may be changed with a sigblock() or sigsetmask() call, or when a signal is delivered to the process.</p> <p>A process may also specify a set of <i>flags</i> for a signal that affect the delivery of that signal.</p> <p>When a signal condition arises for a process, the signal is added to a set of signals pending for the process. If the signal is not currently <i>blocked</i> by the process then it is delivered to the process. When a signal is delivered, the current state of the process is saved, a new signal mask is calculated (as described below), and the signal handler is invoked. The call to the handler is arranged so that if the signal handling routine returns normally the process will resume execution in the context from before the signal's delivery. If the process wishes to resume in a different context, then it must arrange to restore the previous context itself.</p> <p>When a signal is delivered to a process a new signal mask is installed for the duration of the process' signal handler (or until a sigblock() or sigsetmask() call is made). This mask is formed by taking the current signal mask, adding the signal to be delivered, and ORing in the signal mask associated with the handler to be invoked.</p> |

The action to be taken when the signal is delivered is specified by a **sigvec()** structure, which includes the following members:

```
void      (*sv_handler)();          /* signal handler */
int       sv_mask;                  /* signal mask to apply */
int       sv_flags;                 /* see signal options */
#define   SV_ONSTACK                /* take signal on signal stack */
#define   SV_INTERRUPT              /* do not restart system on signal return */
#define   SV_RESETHAND              /* reset handler to SIG_DFL when signal taken*/
```

If the **SV_ONSTACK** bit is set in the flags for that signal, the system will deliver the signal to the process on the signal stack specified with **sigstack(3B)** rather than delivering the signal on the current stack.

If *nvec* is not a **NULL** pointer, **sigvec()** assigns the handler specified by **sv_handler()**, the mask specified by **sv_mask()**, and the flags specified by **sv_flags()** to the specified signal. If *nvec* is a **NULL** pointer, **sigvec()** does not change the handler, mask, or flags for the specified signal.

The mask specified in *nvec* is not allowed to block **SIGKILL**, **SIGSTOP**, or **SIGCONT**. The system enforces this restriction silently.

If *ovec* is not a **NULL** pointer, the handler, mask, and flags in effect for the signal before the call to **sigvec()** are returned to the user. A call to **sigvec()** with *nvec* a **NULL** pointer and *ovec* not a **NULL** pointer can be used to determine the handling information currently in effect for a signal without changing that information.

The following is a list of all signals with names as in the include file `<signal.h>`:

| | |
|-----------------|---|
| SIGHUP | hangup |
| SIGINT | interrupt |
| SIGQUIT* | quit |
| SIGILL* | illegal instruction |
| SIGTRAP* | trace trap |
| SIGABRT* | abort (generated by abort(3C) routine) |
| SIGEMT* | emulator trap |
| SIGFPE* | arithmetic exception |
| SIGKILL | kill (cannot be caught, blocked, or ignored) |

| | |
|-----------|--|
| SIGBUS* | bus error |
| SIGSEGV* | segmentation violation |
| SIGSYS* | bad argument to function |
| SIGPIPE | write on a pipe or other socket with no one to read it |
| SIGALRM | alarm clock |
| SIGTERM | software termination signal |
| SIGURG* | urgent condition present on socket |
| SIGSTOP** | stop (cannot be caught, blocked, or ignored) |
| SIGTSTP** | stop signal generated from keyboard |
| SIGCONT* | continue after stop (cannot be blocked) |
| SIGCHLD* | child status has changed |
| SIGTTIN** | background read attempted from control terminal |
| SIGTTOU** | background write attempted to control terminal |
| SIGIO* | I/O is possible on a descriptor (see <code>fcntl(2)</code>) |
| SIGXCPU | cpu time limit exceeded (see <code>getrlimit(2)</code>) |
| SIGXFSZ | file size limit exceeded (see <code>getrlimit(2)</code>) |
| SIGVTALRM | virtual time alarm; see <code>setitimer()</code> on <code>getitimer(2)</code> |
| SIGPROF | profiling timer alarm; see <code>setitimer()</code> on <code>getitimer(2)</code> |
| SIGWINCH* | window changed (see <code>termio(7I)</code>) |
| SIGLOST | resource lost (see <code>lockd(1M)</code>) |
| SIGUSR1 | user-defined signal 1 |
| SIGUSR2 | user-defined signal 2 |

The starred signals in the list above cause a core image if not caught or ignored.

Once a signal handler is installed, it remains installed until another **sigvec()** call is made, or an **execve(2)** is performed, unless the **SV_RESETHAND** bit is set in the flags for that signal. In that case, the value of the handler for the caught signal will be set to **SIG_DFL** before entering the signal-catching function, unless the signal is **SIGILL**, **SIGPWR**, or **SIGTRAP**. Also, if this bit is set, the bit for that signal in the signal mask will not be set; unless the signal mask associated with that signal blocks that signal, further occurrences of that signal will not be blocked. The **SV_RESETHAND** flag is not available in 4.2BSD, hence it should not be used if backward compatibility is needed.

The default action for a signal may be reinstated by setting the signal's handler to **SIG_DFL**; this default is termination except for signals marked with ***** or ******. Signals marked with ***** are discarded if the action is **SIG_DFL**; signals marked with ****** cause the process to stop. If the process is terminated, a "core image" will be made in the current working directory of the receiving process if the signal is one for which an asterisk appears in the above list (see **core(4)**).

If the handler for that signal is **SIG_IGN**, the signal is subsequently ignored, and pending instances of the signal are discarded.

If a caught signal occurs during certain functions, the call is normally restarted. The call can be forced to terminate prematurely with an **EINTR** error return by setting the **SV_INTERRUPT** bit in the flags for that signal. The **SV_INTERRUPT** flag is not available in 4.2BSD, hence it should not be used if backward compatibility is needed. The affected functions are **read(2)** or **write(2)** on a slow device (such as a terminal or pipe or other socket, but not a file) and during a **wait(2)**.

After a **fork(2)** or **vfork(2)** the child inherits all signals, the signal mask, the signal stack, and the restart/interrupt and reset-signal-handler flags.

The **execve(2)** call resets all caught signals to default action and resets all signals to be caught on the user stack. Ignored signals remain ignored; the signal mask remains the same; signals that interrupt functions continue to do so.

The accuracy of *addr* is machine dependent. For example, certain machines may supply an address that is on the same page as the address that caused the fault. If an appropriate *addr* cannot be computed it will be set to **SIG_NOADDR**.

RETURN VALUES

A 0 value indicates that the call succeeded. A -1 return value indicates that an error occurred and **errno** is set to indicate the reason.

ERRORS

sigvec() will fail and no new signal handler will be installed if one of the following occurs:

EFAULT Either *nvec* or *ovec* is not a **NULL** pointer and points to memory that is not a valid part of the process address space.

EINVAL *sig* is not a valid signal number, or, SIGKILL, or SIGSTOP.

SEE ALSO `intro(2)`, `exec(2)`, `fcntl(2)`, `fork(2)`, `getitimer(2)`, `getrlimit(2)`, `ioctl(2)`, `kill(2)`, `ptrace(2)`, `read(2)`, `umask(2)`, `vfork(2)`, `wait(2)`, `write(2)`, `setjmp(3C)` `sigblock(3B)`, `sigstack(3B)`, `signal(3B)`, `wait(3B)`, `signal(3C)`, `core(4)`, `streamio(7I)`, `termio(7I)`

NOTES Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

SIGPOLL is a synonym for SIGIO. A SIGIO will be issued when a file descriptor corresponding to a STREAMS (see `intro(2)`) file has a “selectable” event pending. Unless that descriptor has been put into asynchronous mode (see `fcntl(2)`), a process may specifically request that this signal be sent using the `I_SETSIG` `ioctl(2)` call (see `streamio(7I)`). Otherwise, the process will never receive SIGPOLLs0.

The handler routine can be declared:

```
void handler(int sig, int code, struct sigcontext *scp, char *addr);
```

Here *sig* is the signal number; *code* is a parameter of certain signals that provides additional detail; *scp* is a pointer to the `sigcontext` structure (defined in `signal.h`), used to restore the context from before the signal; and *addr* is additional address information.

The signals SIGKILL, SIGSTOP, and SIGCONT cannot be ignored.

| | |
|--------------------|--|
| NAME | sigwaitinfo, sigtimedwait – wait for queued signals |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lrt [<i>library</i> ...] #include <signal.h> int sigwaitinfo(const sigset_t * <i>set</i>, siginfo_t * <i>info</i>); int sigtimedwait(const sigset_t * <i>set</i>, siginfo_t * <i>info</i>, const struct timespec * <i>timeout</i>);</pre> |
| DESCRIPTION | <p>The sigwaitinfo() function selects the pending signal from the set specified by <i>set</i> . Should any of multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it will be the lowest numbered one. The selection order between realtime and non-realtime signals, or between multiple pending non-realtime signals, is unspecified. If no signal in <i>set</i> is pending at the time of the call, the calling thread is suspended until one or more signals in <i>set</i> become pending or until it is interrupted by an unblocked, caught signal.</p> <p>The sigwaitinfo() function behaves the same as the sigwait(2) function if the <i>info</i> argument is NULL . If the <i>info</i> argument is non- NULL , the sigwaitinfo() function behaves the same as sigwait(2) , except that the selected signal number is stored in the <i>si_signo</i> member, and the cause of the signal is stored in the <i>si_code</i> member. If any value is queued to the selected signal, the first such queued value is dequeued and, if the <i>info</i> argument is non- NULL , the value is stored in the <i>si_value</i> member of <i>info</i> . The system resource used to queue the signal will be released and made available to queue other signals. If no value is queued, the content of the <i>si_value</i> member is undefined. If no further signals are queued for the selected signal, the pending indication for that signal will be reset. If the value of the <i>si_code</i> member is SI_NOINFO , only the <i>si_signo</i> member of <i>siginfo_t</i> is meaningful, and the value of all other members is unspecified.</p> <p>The sigtimedwait() function behaves the same as sigwaitinfo() except that if none of the signals specified by <i>set</i> are pending, sigtimedwait() waits for the time interval specified in the <i>timespec</i> structure referenced by <i>timeout</i> . If the <i>timespec</i> structure pointed to by <i>timeout</i> is zero-valued and if none of the signals specified by <i>set</i> are pending, then sigtimedwait() returns immediately with an error. If <i>timeout</i> is the NULL pointer, the behavior is unspecified.</p> |

If, while **sigwaitinfo()** or **sigtimedwait()** is waiting, a signal occurs which is eligible for delivery (that is, not blocked by the process signal mask), that signal is handled asynchronously and the wait is interrupted.

RETURN VALUES

Upon successful completion (that is, one of the signals specified by *set* is pending or is generated) **sigwaitinfo()** and **sigtimedwait()** will return the selected signal number. Otherwise, the function returns `-1` and sets `errno` to indicate the error.

ERRORS

The **sigwaitinfo()** and **sigtimedwait()** functions will fail if:

ENOSYS The functions **sigwaitinfo()** and **sigtimedwait()** are not supported by this implementation.

The **sigtimedwait()** function will also fail if:

EAGAIN No signal specified by *set* was generated within the specified timeout period.

The **sigwaitinfo()** and **sigtimedwait()** functions may fail if:

EINTR The wait was interrupted by an unblocked, caught signal. It will be documented in system documentation whether this error will cause these functions to fail.

The **sigtimedwait()** function may also fail if:

EINVAL The *timeout* argument specified a `tv_nsec` value less than zero or greater than or equal to 1000 million. The system only checks for this error if no signal is pending in *set* and it is necessary to wait.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Async-Safe |

SEE ALSO

time(2), **sigqueue(3R)**, **attributes(5)**, **siginfo(5)**, **signal(5)**, **time(5)**

NAME sin – sine function

SYNOPSIS cc [*flag ...*] *file ...* -lm [*library ...*]
#include <math.h>
double **sin**(double *x*);

DESCRIPTION The **sin()** function computes the sine of its argument *x*, measured in radians.

RETURN VALUES Upon successful completion, **sin()** returns the sine of *x*.
If *x* is NaN or ±Inf, NaN is returned.

ERRORS No errors will occur.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **asin(3M)**, **isnan(3M)**, **attributes(5)**

NAME | sinh – hyperbolic sine function

SYNOPSIS | cc [*flag ...*] *file ...* -lm [*library ...*]
 | #include <math.h>
 | double **sinh**(double *x*);

DESCRIPTION | The **sinh()** function computes the hyperbolic sine of *x*.

RETURN VALUES | Upon successful completion, **sinh()** returns the hyperbolic sine of *x*.
 | If the result would cause an overflow, \pm HUGE_VAL is returned and `errno` is set to ERANGE.
 | If *x* is NaN, NaN is returned.
 | For exceptional cases, **matherr**(3M) tabulates the values to be returned as dictated by Standards other than XPG4.

ERRORS | The **sinh()** function will fail if:
 | **ERANGE** the result would cause overflow.

USAGE | An application wishing to check for error situations should set `errno` to 0 before calling **sinh()**. If `errno` is non-zero on return, or the return value is NaN, an error has occurred.

ATTRIBUTES | See **attributes**(5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **asinh**(3M), **cosh**(3M), **isnan**(3M), **matherr**(3M), **tanh**(3M), **attributes**(5), **standards**(5)

| NAME | sleep – suspend execution for interval | | | | |
|--------------------|---|----------------|-----------------|----------|-------------------|
| SYNOPSIS | <pre>/usr/ucb/cc [flag ...] file ...</pre> <pre>int sleep(seconds);</pre> <pre>unsigned seconds;</pre> | | | | |
| DESCRIPTION | <p>sleep() suspends the current process from execution for the number of seconds specified by the argument. The actual suspension time may be up to 1 second less than that requested, because scheduled wakeups occur at fixed 1-second intervals, and may be an arbitrary amount longer because of other activity in the system.</p> <p>sleep() is implemented by setting an interval timer and pausing until it expires. The previous state of this timer is saved and restored. If the sleep time exceeds the time to the expiration of the previous value of the timer, the process sleeps only until the timer would have expired, and the signal which occurs with the expiration of the timer is sent one second later.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Async-Signal-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Async-Signal-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Async-Signal-Safe | | | | |
| SEE ALSO | alarm(2) , getitimer(2) , longjmp(3C) , siglongjmp(3C) , sleep(3C) , usleep(3C) , attributes(5) | | | | |
| NOTES | <p>Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.</p> <p>SIGALRM should <i>not</i> be blocked or ignored during a call to sleep(). Only a prior call to alarm(2) should generate SIGALRM for the calling process during a call to sleep(). A signal-catching function should <i>not</i> interrupt a call to sleep() to call siglongjmp(3C) or longjmp(3C) to restore an environment saved prior to the sleep() call.</p> | | | | |
| WARNINGS | sleep() is slightly incompatible with alarm(2) . Programs that do not execute for at least one second of clock time between successive calls to sleep() indefinitely delay the alarm signal. Use sleep(3C) . Each sleep(3C) call postpones the alarm signal that would have been sent during the requested sleep period to occur one second later. | | | | |

NAME sleep – suspend execution for an interval of time

SYNOPSIS #include <unistd.h>

unsigned int **sleep**(unsigned int *seconds*);

DESCRIPTION

The current process is suspended from execution for the number of *seconds* specified by the argument. The actual suspension time may be less than that requested because any caught signal will terminate the **sleep()** following execution of that signal's catching routine. Also, the suspension time may be longer than requested by an arbitrary amount because of the scheduling of other activity in the system. The value returned by **sleep()** will be the "unslept" amount (the requested time minus the time actually slept) in case the caller had an alarm set to go off earlier than the end of the requested **sleep()** time, or premature arousal because of another caught signal.

The routine is implemented by setting an alarm signal and pausing until it (or some other signal) occurs. The previous state of the alarm signal is saved and restored. The calling program may have set up an alarm signal before calling **sleep()**. If the **sleep()** time exceeds the time until such alarm signal, the process sleeps only until the alarm signal would have occurred. The caller's alarm catch routine is executed just before the **sleep()** routine returns. But if the **sleep()** time is less than the time till such alarm, the prior alarm time is reset to go off at the same time it would have without the intervening **sleep()**.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

alarm(2), **pause(2)**, **signal(3C)**, **attributes(5)**

NOTES

The **SIGALRM** signal should *not* be blocked or ignored during a call to **sleep()**. Only a prior call to **alarm(2)** should generate **SIGALRM** for the calling process during a call to **sleep()**.

In a multithreaded program, only the invoking thread is suspended from execution.

| | |
|-------------------|--|
| NAME | slk_attroff, slk_attr_off, slk_attron, slk_attr_on, slk_attrset, slk_attr_set, slk_clear, slk_color, slk_init, slk_label, slk_noutrefresh, slk_refresh, slk_restore, slk_set, slk_touch, slk_wset – manipulate soft labels |
| SYNOPSIS | <pre>#include <term.h> int slk_attroff(const chtype <i>attrs</i>); int slk_attr_off(const attr_t <i>attrs</i>, void * <i>opts</i>); int slk_attron(const chtype <i>attrs</i>); int slk_attr_on(const attr_t <i>attrs</i>, void * <i>opts</i>); int slk_attrset(const chtype <i>attrs</i>); int slk_attr_set(const attr_t <i>attrs</i>, short <i>color_pair</i>, void * <i>opts</i>); int slk_clear(void); int slk_color(short <i>color_pair</i>); int slk_init(int <i>fmt</i>); char * slk_label(int <i>labnum</i>); int slk_noutrefresh(void); int slk_refresh(void); int slk_restore(void); int slk_set(int <i>labnum</i>, const char * <i>label</i>, int <i>justify</i>); int slk_touch(void); int slk_wset(int <i>labnum</i>, const wchar_t * <i>label</i>, int <i>justify</i>);</pre> |
| PARAMETERS | <p><i>attrs</i> are the foreground window attributes to be added or removed.</p> <p><i>opts</i> Is reserved for future use. Currently, this must be a null pointer.</p> <p><i>color_pair</i> Is a color pair.</p> <p><i>fmt</i> Is the format of how the labels are arranged on the screen.</p> <p><i>labnum</i> Is the number of the soft label.</p> <p><i>label</i> Is the name to be given to a soft label.</p> |

DESCRIPTION

justify Is a number indicating how to justify the label name.

These functions manipulate the soft function-key labels that many terminals feature. For terminals without soft labels, X/Open Curses uses **ripoffline(3XC)** to allocate the bottom line of `stdscr` to emulating them. There can be up to eight soft labels, each with a width of up to eight display columns.

The **slk_init()** function must be called before calling **initscr(3XC)**, **newterm(3XC)**, or **ripoffline()** if you are going to use soft labels. It has the effect of calling **ripoffline()** to reserve a screen line. The `fmt` argument specifies how the labels are to be arranged on the screen. If `fmt` is 0, there is a 3-2-3 arrangement of labels. If `fmt` is 1, there is a 4-4 arrangement.

The **slk_set()** and **slk_wset()** functions assign the label name *label* to the soft label numbered *labnum* (from 1 to 8). *label* can be no more than eight display columns in width. The *justify* argument indicates how the label name is justified within its reserved space:

| | |
|---|------------------------------|
| 0 | Left justify the label name |
| 1 | Center the label name |
| 2 | Right justify the label name |

The **slk_refresh()** and **slk_noutrefresh()** functions correspond to the **wrefresh(3XC)** and **wnoutrefresh(3XC)** functions described in the **doupdate(3XC)** man page and are used to update the actual soft label text on the screen.

The **slk_label()** returns the label name assigned to the label number *labnum*.

The **slk_clear()** clears the soft labels from the screen.

The **slk_restore()** restores the soft label information to the screen after a call to **slk_clear()**.

The **slk_touch()** marks all soft labels as needing to be updated when **slk_refresh()** or **slk_noutrefresh()** is next called.

The **slk_attron()**, **slk_attrset()**, and **slk_attroff()** functions behave similarly to the **attron(3XC)**, **attrset(3XC)**, and **attroff(3XC)** functions.

The **slk_attr_on()**, **slk_attr_off()**, **slk_attr_set()** and **slk_color()** functions behave similarly to the **attr_on(3XC)**, **attr_off(3XC)**, **attr_set(3XC)**, and **color_set(3XC)** functions. As a result, they support color and the attribute constants whose name begin with `WA_`.

RETURN VALUES

On success, the **slk_label()** function returns the requested label name. Otherwise, it returns a null pointer.

On success, the other functions return OK . Otherwise, they return ERR .

ERRORS

None.

SEE ALSO

attr_get(3XC) , **attroff(3XC)** , **delscreen(3XC)** , **ripoffline(3XC)**

| | |
|--------------------|--|
| NAME | socket – create an endpoint for communication |
| SYNOPSIS | <pre>cc [flag ...] file ... -lsocket -lnsl [library ...] #include <sys/types.h> #include <sys/socket.h></pre> <pre>int socket(int domain, int type, int protocol);</pre> |
| DESCRIPTION | <p>socket() creates an endpoint for communication and returns a descriptor.</p> <p>The <i>domain</i> parameter specifies a communications domain within which communication will take place; this selects the protocol family which should be used. The protocol family generally is the same as the address family for the addresses supplied in later operations on the socket. These families are defined in the include file <code><sys/socket.h></code>. There must be an entry in the <code>netconfig(4)</code> file for at least each protocol family and type required. If <i>protocol</i> has been specified, but no exact match for the tuple family, type, protocol is found, then the first entry containing the specified family and type with zero for protocol will be used. The currently understood formats are:</p> <pre>PF_UNIX UNIX system internal protocols</pre> <pre>PF_INET ARPA Internet protocols</pre> <p>The socket has the indicated <i>type</i>, which specifies the communication semantics. Currently defined types are:</p> <pre>SOCK_STREAM SOCK_DGRAM SOCK_RAW SOCK_SEQPACKET SOCK_RDM</pre> <p>A <code>SOCK_STREAM</code> type provides sequenced, reliable, two-way connection-based byte streams. An out-of-band data transmission mechanism may be supported. A <code>SOCK_DGRAM</code> socket supports datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length). A <code>SOCK_SEQPACKET</code> socket may provide a sequenced, reliable, two-way connection-based data transmission path for datagrams of fixed maximum length; a consumer may be required to read an entire packet with each read system call. This facility is protocol specific, and presently not implemented for any protocol family. <code>SOCK_RAW</code> sockets provide access to internal network interfaces. The types <code>SOCK_RAW</code>, which is available only to the super-user, and <code>SOCK_RDM</code>, for which no implementation currently exists, are not described here.</p> <p><i>protocol</i> specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family. However, multiple protocols may exist, in which case a</p> |

particular protocol must be specified in this manner. The protocol number to use is particular to the “communication domain” in which communication is to take place. If a protocol is specified by the caller, then it will be packaged into a socket level option request and sent to the underlying protocol layers.

Sockets of type `SOCK_STREAM` are full-duplex byte streams, similar to pipes. A stream socket must be in a *connected* state before any data may be sent or received on it. A connection to another socket is created with a `connect(3N)` call. Once connected, data may be transferred using `read(2)` and `write(2)` calls or some variant of the `send(3N)` and `recv(3N)` calls. When a session has been completed, a `close(2)` may be performed. Out-of-band data may also be transmitted as described on the `send(3N)` manual page and received as described on the `recv(3N)` manual page.

The communications protocols used to implement a `SOCK_STREAM` insure that data is not lost or duplicated. If a piece of data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, then the connection is considered broken and calls will indicate an error with `-1` returns and with `ETIMEDOUT` as the specific code in the global variable `errno`. The protocols optionally keep sockets “warm” by forcing transmissions roughly every minute in the absence of other activity. An error is then indicated if no response can be elicited on an otherwise idle connection for an extended period (for instance 5 minutes). A `SIGPIPE` signal is raised if a process sends on a broken stream; this causes naive processes, which do not handle the signal, to exit.

`SOCK_SEQPACKET` sockets employ the same system calls as `SOCK_STREAM` sockets. The only difference is that `read(2)` calls will return only the amount of data requested, and any remaining in the arriving packet will be discarded.

`SOCK_DGRAM` and `SOCK_RAW` sockets allow datagrams to be sent to correspondents named in `sendto(3N)` calls. Datagrams are generally received with `recvfrom(3N)`, which returns the next datagram with its return address.

An `fcntl(2)` call can be used to specify a process group to receive a `SIGURG` signal when the out-of-band data arrives. It may also enable non-blocking I/O and asynchronous notification of I/O events with `SIGIO` signals.

The operation of sockets is controlled by socket level *options*. These options are defined in the file `<sys/socket.h>`. `setsockopt(3N)` and `getsockopt(3N)` are used to set and get options, respectively.

RETURN VALUES

A `-1` is returned if an error occurs. Otherwise the return value is a descriptor referencing the socket.

ERRORS

The `socket()` call fails if:

| | |
|------------------------|--|
| EACCES | Permission to create a socket of the specified type and/or protocol is denied. |
| EMFILE | The per-process descriptor table is full. |
| ENOMEM | Insufficient user memory is available. |
| ENOSR | There were insufficient STREAMS resources available to complete the operation. |
| EPROTONOSUPPORT | The protocol type or the specified protocol is not supported within this domain. |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

`close(2)`, `fcntl(2)`, `ioctl(2)`, `read(2)`, `write(2)`, `accept(3N)`, `bind(3N)`, `connect(3N)`, `getsockname(3N)`, `getsockopt(3N)`, `listen(3N)`, `recv(3N)`, `setsockopt(3N)`, `send(3N)`, `shutdown(3N)`, `socketpair(3N)`, `attributes(5)`, `in(5)`, `socket(5)`

| | |
|--------------------|---|
| NAME | socket – create an endpoint for communication |
| SYNOPSIS | <pre>cc [<i>flag ...</i>] <i>file ...</i> -lxnet [<i>library ...</i>] #include <sys/socket.h></pre> <pre>int socket(int <i>domain</i>, int <i>type</i>, int <i>protocol</i>);</pre> |
| DESCRIPTION | <p>The socket() function creates an unbound socket in a communications domain, and returns a file descriptor that can be used in later function calls that operate on sockets.</p> <p>The function takes the following arguments:</p> <p>domain Specifies the communications domain in which a socket is to be created.</p> <p>type Specifies the type of socket to be created.</p> <p>protocol Specifies a particular protocol to be used with the socket. Specifying a <i>protocol</i> of 0 causes socket() to use an unspecified default protocol appropriate for the requested socket type.</p> <p>The <i>domain</i> argument specifies the address family used in the communications domain. The address families supported by the system are implementation-dependent.</p> <p>The <i>sys/socket.h</i> header defines at least the following values for the <i>domain</i> argument:</p> <p>AF_UNIX File system pathnames.</p> <p>AF_INET Internet address.</p> <p>The <i>type</i> argument specifies the socket type, which determines the semantics of communication over the socket. The socket types supported by the system are implementation-dependent. Possible socket types include:</p> <p>SOCK_STREAM Provides sequenced, reliable, bidirectional, connection-mode byte streams, and may provide a transmission mechanism for out-of-band data.</p> <p>SOCK_DGRAM Provides datagrams, which are connectionless-mode, unreliable messages of fixed maximum length.</p> <p>SOCK_SEQPACKET Provides sequenced, reliable, bidirectional, connection-mode transmission path for records. A record can be sent using one or more output operations and received using one or more input</p> |

operations, but a single operation never transfers part of more than one record. Record boundaries are visible to the receiver via the MSG_EOR flag.

If the *protocol* argument is non-zero, it must specify a protocol that is supported by the address family. The protocols supported by the system are implementation-dependent.

The process may need to have appropriate privileges to use the **socket()** function or to create some sockets.

USAGE

The documentation for specific address families specify which protocols each address family supports. The documentation for specific protocols specify which socket types each protocol supports.

The application can determine if an address family is supported by trying to create a socket with *domain* set to the protocol in question.

RETURN VALUES

Upon successful completion, **socket()** returns a nonnegative integer, the socket file descriptor. Otherwise a value of -1 is returned and `errno` is set to indicate the error.

ERRORS

The **socket()** function will fail if:

| | |
|---|--|
| EAFNOSUPPORT | The implementation does not support the specified address family. |
| EMFILE | No more file descriptors are available for this process. |
| ENFILE | No more file descriptors are available for the system. |
| EPROTONOSUPPORT | The protocol is not supported by the address family, or the protocol is not supported by the implementation. |
| EPROTOTYPE | The socket type is not supported by the protocol. |
| The socket() function may fail if: | |
| EACCES | The process does not have appropriate privileges. |
| ENOBUFS | Insufficient resources were available in the system to perform the operation. |
| ENOMEM | Insufficient memory was available to fulfill the request. |

ENOSR

There were insufficient STREAMS resources available for the operation to complete.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

accept(3XN), **bind(3XN)**, **connect(3XN)**, **getsockname(3XN)**, **getsockopt(3XN)**, **listen(3XN)**, **recv(3XN)**, **recvfrom(3XN)**, **recvmsg(3XN)**, **send(3XN)**, **sendmsg(3XN)**, **setsockopt(3XN)**, **shutdown(3XN)**, **socketpair(3XN)**, **attributes(5)**

| NAME | socketpair – create a pair of connected sockets | | | | | | | | | | | | |
|------------------------|--|---------------------|--|---------------|--|---------------|---|--------------|--|---------------------|---|------------------------|--|
| SYNOPSIS | <pre>cc [flag ...] file ... -lsocket -lnsl [library ...] #include <sys/types.h> #include <sys/socket.h></pre> <p>int socketpair(int domain, int type, int protocol, int sv[2]);</p> | | | | | | | | | | | | |
| DESCRIPTION | The socketpair() library call creates an unnamed pair of connected sockets in the specified address family <i>d</i> , of the specified <code>type</code> , and using the optionally specified <i>protocol</i> . The descriptors used in referencing the new sockets are returned in <code>sv[0]</code> and <code>sv[1]</code> . The two sockets are indistinguishable. | | | | | | | | | | | | |
| RETURN VALUES | socketpair() returns <code>-1</code> on failure, and <code>0</code> on success. | | | | | | | | | | | | |
| ERRORS | <p>The call succeeds unless:</p> <table border="0"> <tr> <td style="vertical-align: top;">EAFNOSUPPORT</td> <td>The specified address family is not supported on this machine.</td> </tr> <tr> <td style="vertical-align: top;">EMFILE</td> <td>Too many descriptors are in use by this process.</td> </tr> <tr> <td style="vertical-align: top;">ENOMEM</td> <td>There was insufficient user memory for the operation to complete.</td> </tr> <tr> <td style="vertical-align: top;">ENOSR</td> <td>There were insufficient STREAMS resources for the operation to complete.</td> </tr> <tr> <td style="vertical-align: top;">EOPNOSUPPORT</td> <td>The specified protocol does not support creation of socket pairs.</td> </tr> <tr> <td style="vertical-align: top;">EPROTONOSUPPORT</td> <td>The specified protocol is not supported on this machine.</td> </tr> </table> | EAFNOSUPPORT | The specified address family is not supported on this machine. | EMFILE | Too many descriptors are in use by this process. | ENOMEM | There was insufficient user memory for the operation to complete. | ENOSR | There were insufficient STREAMS resources for the operation to complete. | EOPNOSUPPORT | The specified protocol does not support creation of socket pairs. | EPROTONOSUPPORT | The specified protocol is not supported on this machine. |
| EAFNOSUPPORT | The specified address family is not supported on this machine. | | | | | | | | | | | | |
| EMFILE | Too many descriptors are in use by this process. | | | | | | | | | | | | |
| ENOMEM | There was insufficient user memory for the operation to complete. | | | | | | | | | | | | |
| ENOSR | There were insufficient STREAMS resources for the operation to complete. | | | | | | | | | | | | |
| EOPNOSUPPORT | The specified protocol does not support creation of socket pairs. | | | | | | | | | | | | |
| EPROTONOSUPPORT | The specified protocol is not supported on this machine. | | | | | | | | | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe | | | | | | | | |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | | | | | | | |
| MT-Level | Safe | | | | | | | | | | | | |
| SEE ALSO | pipe(2) , read(2) , write(2) , attributes(5) , socket(5) | | | | | | | | | | | | |
| NOTES | This call is currently implemented only for the <code>AF_UNIX</code> address family. | | | | | | | | | | | | |

| | |
|--------------------|---|
| NAME | socketpair – create a pair of connected sockets |
| SYNOPSIS | <pre>cc [flag ...] file ... -lxnet [library ...] #include <sys/socket.h></pre> <pre>int socketpair(int domain, int type, int protocol, int socket_vector[2]);</pre> |
| DESCRIPTION | <p>The socketpair() function creates an unbound pair of connected sockets in a specified <i>domain</i>, of a specified <i>type</i>, under the protocol optionally specified by the <i>protocol</i> argument. The two sockets are identical. The file descriptors used in referencing the created sockets are returned in <i>socket_vector</i>0 and <i>socket_vector</i>1.</p> <p>domain Specifies the communications domain in which the sockets are to be created.</p> <p>type Specifies the type of sockets to be created.</p> <p>protocol Specifies a particular protocol to be used with the sockets. Specifying a <i>protocol</i> of 0 causes socketpair() to use an unspecified default protocol appropriate for the requested socket type.</p> <p>socket_vector Specifies a 2-integer array to hold the file descriptors of the created socket pair.</p> <p>The <i>type</i> argument specifies the socket type, which determines the semantics of communications over the socket. The socket types supported by the system are implementation-dependent. Possible socket types include:</p> <p>SOCK_STREAM Provides sequenced, reliable, bidirectional, connection-mode byte streams, and may provide a transmission mechanism for out-of-band data.</p> <p>SOCK_DGRAM Provides datagrams, which are connectionless-mode, unreliable messages of fixed maximum length.</p> <p>SOCK_SEQPACKET Provides sequenced, reliable, bidirectional, connection-mode transmission path for records. A record can be sent using one or more output operations and received using one or more input operations, but a single operation never transfers part of more than one record. Record boundaries are visible to the receiver via the MSG_EOR flag.</p> <p>If the <i>protocol</i> argument is non-zero, it must specify a protocol that is supported by the address family. The protocols supported by the system are implementation-dependent.</p> |

| | |
|----------------------|--|
| | The process may need to have appropriate privileges to use the socketpair() function or to create some sockets. |
| USAGE | <p>The documentation for specific address families specifies which protocols each address family supports. The documentation for specific protocols specifies which socket types each protocol supports.</p> <p>The socketpair() function is used primarily with UNIX domain sockets and need not be supported for other domains.</p> |
| RETURN VALUES | Upon successful completion, this function returns 0. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error. |
| ERRORS | <p>The socketpair() function will fail if:</p> <p>EAFNOSUPPORT The implementation does not support the specified address family.</p> <p>EMFILE No more file descriptors are available for this process.</p> <p>ENFILE No more file descriptors are available for the system.</p> <p>EOPNOTSUPP The specified protocol does not permit creation of socket pairs.</p> <p>EPROTONOSUPPORT The protocol is not supported by the address family, or the protocol is not supported by the implementation.</p> <p>EPROTOTYPE The socket type is not supported by the protocol.</p> <p>The socketpair() function may fail if:</p> <p>EACCES The process does not have appropriate privileges.</p> <p>ENOBUFS Insufficient resources were available in the system to perform the operation.</p> <p>ENOMEM Insufficient memory was available to fulfill the request.</p> <p>ENOSR There were insufficient STREAMS resources available for the operation to complete.</p> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

socketpair(3XN)

X/Open Networking Services Library Functions

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO `socket(3XN)`, `attributes(5)`

| | | | | | | | |
|--------------------|---|-----------------|------------------------------------|-----------------|----------------------------------|---------------|---|
| NAME | spray - scatter data in order to test the network | | | | | | |
| SYNOPSIS | <pre>cc [flag ...] file ... -lsocket -lnsl [library ...] #include <rpcsvc/spray.h></pre> <pre>bool_t xdr_sprayarr(XDR *xdrs, sprayarr *objp);</pre> <pre>bool_t xdr_spraycumul(XDR *xdrs, spraycumul *objp);</pre> | | | | | | |
| DESCRIPTION | <p>The spray program sends packets to a given machine to test communications with that machine.</p> <p>The spray program is not a C function interface, per se, but it can be accessed using the generic remote procedure calling interface <code>clnt_call()</code>. See <code>rpc_clnt_calls(3N)</code>. The program sends a packet to the called host. The host acknowledges receipt of the packet. The program counts the number of acknowledgments and can return that count.</p> <p>The spray program currently supports the following procedures, which should be called in the order given:</p> <table border="0"> <tr> <td style="padding-right: 20px;">SPRAYPROC_CLEAR</td> <td>This procedure clears the counter.</td> </tr> <tr> <td>SPRAYPROC_SPRAY</td> <td>This procedure sends the packet.</td> </tr> <tr> <td>SPRAYPROC_GET</td> <td>This procedure returns the count and the amount of time since the last SPRAYPROC_CLEAR.</td> </tr> </table> | SPRAYPROC_CLEAR | This procedure clears the counter. | SPRAYPROC_SPRAY | This procedure sends the packet. | SPRAYPROC_GET | This procedure returns the count and the amount of time since the last SPRAYPROC_CLEAR. |
| SPRAYPROC_CLEAR | This procedure clears the counter. | | | | | | |
| SPRAYPROC_SPRAY | This procedure sends the packet. | | | | | | |
| SPRAYPROC_GET | This procedure returns the count and the amount of time since the last SPRAYPROC_CLEAR. | | | | | | |
| EXAMPLES | <p>EXAMPLE 1 Using spray()</p> <p>The following code fragment demonstrates how the spray program is used:</p> <pre>#include <rpc/rpc.h> #include <rpcsvc/spray.h> . . . spraycumul spray_result; sprayarr spray_data; char buf[100]; /* arbitrary data */ int loop = 1000; CLIENT *clnt; struct timeval timeout0 = {0, 0}; struct timeval timeout25 = {25, 0}; spray_data.sprayarr_len = (uint_t)100; spray_data.sprayarr_val = buf; clnt = clnt_create("somehost", SPRAYPROC, SPRAYVERS, "netpath"); if (clnt == (CLIENT *)NULL) { /* handle this error */ } if (clnt_call(clnt, SPRAYPROC_CLEAR, xdr_void, NULL, xdr_void, NULL, timeout25)) { /* handle this error */ }</pre> | | | | | | |

```

}
while (loop- > 0) {
  if (clnt_call(clnt, SPRAYPROC_SPRAY,
    xdr_sprayarr, &spray_data, xdr_void, NULL, timeout0)) {
    /* handle this error */
  }
}
if (clnt_call(clnt, SPRAYPROC_GET,
  xdr_void, NULL, xdr_spraycumul, &spray_result, timeout25)) {
  /* handle this error */
}
printf("Acknowledged %ld of 1000 packets in %d secs %d usecs\n",
  spray_result.counter,
  spray_result.clock.sec,
  spray_result.clock.usec);

```

ATTRIBUTES

See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

[spray\(1M\)](#), [rpc_clnt_calls\(3N\)](#), [attributes\(5\)](#)

NOTES

This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

A spray program is not useful as a networking benchmark as it uses unreliable connectionless transports, for example, udp. It can report a large number of packets dropped, when the drops were caused by the program sending packets faster than they can be buffered locally, that is, before the packets get to the network medium.

NAME sqrt – square root function

SYNOPSIS cc [*flag ...*] *file ...* -lm [*library ...*]
#include <math.h>
double sqrt(double x);

DESCRIPTION The **sqrt()** function computes the square root of *x*.

RETURN VALUES Upon successful completion, **sqrt()** returns the square root of *x*.
If *x* is NaN, NaN is returned.
If *x* is negative, NaN is returned and `errno` is set to EDOM.

ERRORS The **sqrt()** function will fail if:
EDOM The value of *x* is negative.

USAGE An application wishing to check for error situations should set `errno` to 0 before calling **sqrt()**. If `errno` is non-zero on return, or the return value is NaN, an error has occurred.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **isnan(3M)**, **attributes(5)**

| | |
|--------------------|---|
| NAME | SSAAgentIsAlive, SSAGetTrapPort, SSARegSubtable, SSARegSubagent, SSARegSubtree, SSASendTrap, SSASubagentOpen – Sun Solstice Enterprise Agent registration and communication helper functions |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lssagent -lssasmp [<i>library</i> ..] #include <impl.h> extern int SSAAgentIsAlive(IPAddress * <i>agent_addr</i>, int * <i>port</i>, char * <i>community</i>, struct timeval * <i>timeout</i>); extern int SSAGetTrapPort(); extern int * SSARegSubagent(Agent* <i>agent</i>); int SSARegSubtable(SSA_Table * <i>table</i>); int SSARegSubtree(SSA_Subtree * <i>subtree</i>); extern void SSASendTrap(char * <i>name</i>); extern int SSASubagentOpen(int * <i>num_of_retry</i>, char * <i>agent_name</i>);</pre> |
| DESCRIPTION | <p>The SSAAgentIsAlive() function returns <code>TRUE</code> if the master agent is alive, otherwise returns <code>FALSE</code>. The <i>agent_addr</i> parameter is the address of the agent. Specify the security token in the <i>community</i> parameter. You can specify the maximum amount of time to wait for a response with the <i>timeout</i> parameter.</p> <p>The SSAGetTrapPort() function returns the port number used by the Master Agent to communicate with the subagent.</p> <p>The SSARegSubagent() function enables a subagent to register and unregister with a Master Agent. The <i>agent</i> parameter is a pointer to an <code>Agent</code> structure containing the following members:</p> <pre>int timeout; \011/* optional */ \011</pre> |

```

int     agent_id;      /* required */ \011
int     agent_status; /* required */ \011
char    *personal_file;\011/* optional */ \011
char    *config_file; /* optional */ \011
char    *executable;  /* optional */ \011
char    *version_string;\011/* optional */ \011
char    *protocol;    /* optional */ \011
int     process_id;   /* optional */ \011
char    *name;        /* optional */ \011
int     system_up_time; /* optional */ \011
int     watch_dog_time; /* optional */ \011
Address address;      /* required */ \011
struct  _Agent;       /* reserved */ \011
struct  _Subtree;     /* reserved */

```

The `agent_id` member is an integer value returned by the **SSASubagentOpen()** function. After calling **SSASubagentOpen()**, you pass the `agent_id` in the **SSARegSubagent()** call to register the subagent with the Master Agent.

The following values are supported for `agent_status`:

```

SSA_OPER_STATUS_ACTIVE
SSA_OPER_STATUS_NOT_IN_SERVICE
SSA_OPER_STATUS_DESTROY

```

You pass `SSA_OPER_STATUS_DESTROY` as the value in a **SSARegSubagent()** function call when you want to unregister the agent from the Master Agent.

`Address` has the same structure as `sockaddr_in`, that is a common UNIX structure containing the following members:

```

\011short\011    sin_family;
\011ushort_t\011 sin_port;
\011struct\011   in_addr sin_addr;
\011char\011     sin_zero[8];

```

The **SSARegSubtable()** function registers a MIB table with the Master Agent. If this function is successful, an index number is returned, otherwise 0 is returned. The `table` parameter is a pointer to a `SSA_Table` structure containing the following members:

```

\011int\011regTblIndex;\011      /* index value */
\011int\011regTblAgentID;\011     /* current agent ID */
\011Oid\011regTblOID;\011       /* Object ID of the table */

```

```

\011int\011regTblStartColumn;\011 /* start column index */
\011int\011regTblEndColumn;\011  /* end column index */
\011int\011regTblStartRow;\011   /* start row index */
\011int\011regTblEndRow;\011     /* end row index */
\011int\011regTblStatus;\011     /* status */

```

The `regTblStatus` can have one of the following values:

```

SSA_OPER_STATUS_ACTIVE
SSA_OPER_STATUS_NOT_IN_SERVICE

```

The **SSARegSubtree()** function registers a MIB subtree with the master agent. If successful this function returns an index number, otherwise 0 is returned. The *subtree* parameter is a pointer to a `SSA_Subtree` structure containing the following members:

```

\011int\011regTreeIndex;\011     /* index value */
\011int\011regTreeAgentID;\011   /* current agent ID */
\011Oid\011name;\011             /* Object ID to register */
\011int\011regtreeStatus;\011    /* status */

```

The `regtreeStatus` can have one of the following values:

```

SSA_OPER_STATUS_ACTIVE
SSA_OPER_STATUS_NOT_IN_SERVICE

```

The **SSASendTrap()** function instructs the Master Agent to send a trap notification, based on the keyword passed with *name*. When your subagent MIB is compiled by `mibcodegen`, it creates a lookup table of the trap notifications defined in the MIB. By passing the name of the trap notification type as *name*, the subagent instructs the Master Agent to construct the type of trap defined in the MIB.

The **SSASubagentOpen()** function initializes communication between the subagent and the Master Agent. You must call this function before calling **SSARegSubagent()** to register the subagent with the Master Agent. The **SSASubagentOpen()** function returns a unique agent ID that is passed in the **SSARegSubagent()** call to register the subagent. If 0 is returned as the agent ID, the attempt to initialize communication with the Master Agent was unsuccessful. Since UDP is used to initialize communication with the Master Agent, you may want to set the value of *num_of_retry* to make multiple attempts.

The value for *agent_name* must be unique within the domain for which the Master Agent is responsible.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

attributes(5)

| | |
|--------------------|---|
| NAME | SSAOidCmp, SSAOidCpy, SSAOidDup, SSAOidFree, SSAOidInit, SSAOidNew, SSAOidString, SSAOidStrToOid, SSAOidZero – Sun Solstice Enterprise Agent OID helper functions |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ... -lssasnmp [<i>library</i> ..] #include <impl.h> int SSAOidCmp(Oid * <i>oid1</i>, Oid * <i>oid2</i>); int SSAOidCpy(Oid * <i>oid1</i>, Oid * <i>oid2</i>, char * <i>error_label</i>); Oid * SSAOidDup(Oid * <i>oid</i>, char * <i>error_label</i>); void SSAOidFree(Oid * <i>oid</i>); int SSAOidInit(Oid * <i>oid</i>, Subid * <i>subids</i>, int <i>len</i>, char * <i>error_label</i>); Oid * SSAOidNew(); char * SSAOidString(Oid * <i>oid</i>); Oid * SSAOidStrToOid(char* <i>name</i>, char * <i>error_label</i>); void SSAOidZero(Oid * <i>oid</i>); </pre> |
| DESCRIPTION | <p>The SSAOidCmp() function performs a comparison of the given OIDs. This function returns:</p> <ul style="list-style-type: none"> 0 if <i>oid1</i> is equal to <i>oid2</i> 1 if <i>oid1</i> is greater than <i>oid2</i> -1 if <i>oid1</i> is less than <i>oid2</i> <p>The SSAOidCpy() function makes a deep copy of <i>oid2</i> to <i>oid1</i> . This function assumes <i>oid1</i> has been processed by the SSAOidZero() function. Memory is allocated inside <i>oid1</i> and the contents of <i>oid2</i> , not just the pointer, is copied to <i>oid1</i> . If an error is encountered, an error message is stored in the <i>error_label</i> buffer.</p> <p>The SSAOidDup() function returns a clone of <i>oid</i> , by using the deep copy. Error information is stored in the <i>error_label</i> buffer.</p> |

The **SSAOidFree()** function frees the OID instance, with its content.

The **SSAOidNew()** function returns a new OID.

The **SSAOidInit()** function copies the Subid array from *subids* to the OID instance with the specified length *len*. This function assumes that the OID instance has been processed by the **SSAOidZero()** function or no memory is allocated inside the OID instance. If an error is encountered, an error message is stored in the *error_label* buffer.

The **SSAOidString()** function returns a char pointer for the printable form of the given *oid*.

The **SSAOidStrToOid()** function returns a new OID instance from *name*. If an error is encountered, an error message is stored in the *error_label* buffer.

The **SSAOidZero()** function frees the memory used by the OID object for buffers, but not the OID instance itself.

RETURN VALUES

The **SSAOidNew()** and **SSAOidStrToOid()** functions return 0 if an error is detected.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO

attributes(5)

| | |
|----------------------|--|
| NAME | SSAStringCpy, SSAStringInit, SSAStringToChar, SSAStringZero – Sun Solstice Enterprise Agent string helper functions |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lssasnmp [<i>library</i> ..] #include <impl.h> void * SSAStringZero(String * <i>string</i>); int SSAStringInit(String * <i>string</i>, uchar_t * <i>chars</i>, int <i>len</i>, char * <i>error_label</i>); int SSAStringCpy(String * <i>string1</i>, String * <i>string2</i>, char * <i>error_label</i>); char * SSAStringToChar(String <i>string</i>);</pre> |
| DESCRIPTION | <p>The SSAStringCpy() function makes a deep copy of <i>string2</i> to <i>string1</i> . This function assumes that <i>string1</i> has been processed by the SSAStringZero() function. Memory is allocated inside the <i>string1</i> and the contents of <i>string2</i> , not just the pointer, is copied to the <i>string1</i> . If an error is encountered, an error message is stored in the <i>error_label</i> buffer.</p> <p>The SSAStringInit() function copies the char array from <i>chars</i> to the string instance with the specified length <i>len</i> . This function assumes that the string instance has been processed by the SSAStringZero() function or no memory is allocated inside the string instance. If an error is encountered, an error message is stored in the <i>error_label</i> buffer.</p> <p>The SSAStringToChar() function returns a temporary char array buffer for printing purposes.</p> <p>The SSAStringZero() function frees the memory inside of the String instance, but not the string object itself.</p> |
| RETURN VALUES | The SSAStringInit() and SSAStringCpy() functions return 0 if successful and -1 if error. |
| ATTRIBUTES | See attributes (5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO**attributes(5)**

NAME | signal, gsignal – software signals

SYNOPSIS | #include <signal.h>
 | void(*ssignal (int sig, int (* action)(int)))(int);
 | int gsignal(int sig);

DESCRIPTION | The **ssignal()** and **gsignal()** functions implement a software facility similar to **signal(3C)** . This facility is made available to users for their own purposes.

ssignal() | Software signals made available to users are associated with integers in the inclusive range 1 through 17. A call to **ssignal()** associates a procedure, *action* , with the software signal *sig* ; the software signal, *sig* , is raised by a call to **gsignal()** . Raising a software signal causes the action established for that signal to be taken.

| The first argument to **ssignal()** is a number identifying the type of signal for which an action is to be established. The second argument defines the action; it is either the name of a (user-defined) *action function* or one of the manifest constants SIG_DFL (default) or SIG_IGN (ignore). The **ssignal()** function returns the action previously established for that signal type; if no action has been established or the signal number is illegal, **ssignal()** returns SIG_DFL .

gsignal() | The **gsignal()** raises the signal identified by its argument, *sig* .

| If an action function has been established for *sig* , then that action is reset to SIG_DFL and the action function is entered with argument *sig* . The **gsignal()** function returns the value returned to it by the action function.

| If the action for *sig* is SIG_IGN , **gsignal()** returns the value 1 and takes no other action.

| If the action for *sig* is SIG_DFL , **gsignal()** returns the value 0 and takes no other action.

| If *sig* has an illegal value or no action was ever specified for *sig* , **gsignal()** returns the value 0 and takes no other action.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Unsafe |

SEE ALSO | **raise(3C)** , **signal(3C)** , **attributes(5)**

| | |
|----------------------|--|
| NAME | standend, standout, wstandend, wstandout – set/clear window attributes |
| SYNOPSIS | <pre>#include <curses.h> int standend(void); int standout(void); int wstandend(WINDOW * win); int wstandout(WINDOW * win);</pre> |
| PARAMETERS | <p><i>win</i> Is a pointer to the window in which attribute changes are to be made.</p> |
| DESCRIPTION | <p>The standend() and wstandend() functions turn off all attributes associated with <code>stdscr</code> and <i>win</i> respectively.</p> <p>The standout() and wstandout() functions turn on the <code>A_STANDOUT</code> attribute of <code>stdscr</code> and <i>win</i> respectively.</p> |
| RETURN VALUES | These functions always return 1. |
| ERRORS | None. |
| SEE ALSO | <code>attr_get(3XC)</code> , <code>attroff(3XC)</code> |

| | |
|--------------------|---|
| NAME | stdio – standard buffered input/output package |
| SYNOPSIS | <pre>#include <stdio.h> extern FILE *stdin; extern FILE *stdout; extern FILE *stderr;</pre> |
| DESCRIPTION | <p>The functions described in the entries of section 3S of this manual constitute an efficient, user-level I/O buffering scheme. The in-line macros getc() and putc() handle characters quickly. The macros getchar(3S) and putchar(3S), and the higher-level routines fgetc(3S), fgets(3S), fprintf(3S), fputc(3S), fputs(3S), fread(3S), fscanf(3S), fwrite(3S), gets(3S), getw(3S), printf(3S), puts(3S), putw(3S), and scanf(3S) all use or act as if they use getc() and putc(); they can be freely intermixed.</p> <p>A file with associated buffering is called a <i>stream</i> (see intro(3)) and is declared to be a pointer to a defined type <code>FILE</code>. The fopen(3S) function creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. Normally, there are three open streams with constant pointers declared in the <code><stdio.h></code> header and associated with the standard open files:</p> <pre>stdin standard input file stdout standard output file stderr standard error file</pre> <p>The following symbolic values in <code><unistd.h></code> define the file descriptors that will be associated with the C-language <code>stdin</code>, <code>stdout</code> and <code>stderr</code> when the application is started:</p> <pre>STDIN_FILENO Standard input value 0 stdin STDOUT_FILENO Standard output value 1 stdout STDERR_FILENO Standard error value 2 stderr</pre> <p>The constant <code>NULL</code> designates a null pointer.</p> <p>The integer-constant <code>EOF</code> is returned upon end-of-file or error by most integer functions that deal with streams (see the individual descriptions for details).</p> <p>The integer constant <code>BUFSIZ</code> specifies the size of the buffers used by the particular implementation.</p> |

The integer constant `FILENAME_MAX` specifies the number of bytes needed to hold the longest pathname of a file allowed by the implementation. If the system does not impose a maximum limit, this value is the recommended size for a buffer intended to hold a file's pathname.

The integer constant `FOPEN_MAX` specifies the minimum number of files that the implementation guarantees can be open simultaneously. Note that no more than 255 files may be opened using `fopen()`, and only file descriptors 0 through 255 can be used in a stream.

The functions and constants mentioned in the entries of section 3S of this manual are declared in that header and need no further declaration. The constants and the following "functions" are implemented as macros (redeclaration of these names is perilous): `getc()`, `getchar()`, `putc()`, `putchar()`, `ferror(3S)`, `feof(3S)`, `clearerr(3S)`, and `fileno(3S)`. There are also function versions of `getc()`, `getchar()`, `putc()`, `putchar()`, `ferror()`, `feof()`, `clearerr()`, and `fileno()`.

Output streams, with the exception of the standard error stream `stderr`, are by default buffered if the output refers to a file and line-buffered if the output refers to a terminal. The standard error output stream `stderr` is by default unbuffered, but use of `freopen()` (see `fopen(3S)`) will cause it to become buffered or line-buffered. When an output stream is unbuffered, information is queued for writing on the destination file or terminal as soon as written; when it is buffered, many characters are saved up and written as a block. When it is line-buffered, each line of output is queued for writing on the destination terminal as soon as the line is completed (that is, as soon as a new-line character is written or terminal input is requested). The `setbuf()` or `setvbuf()` functions (both described on the `setbuf(3S)` manual page) may be used to change the stream's buffering strategy.

Interactions of Other FILE-Type C Functions

A single open file description can be accessed both through streams and through file descriptors. Either a file descriptor or a stream will be called a *handle* on the open file description to which it refers; an open file description may have several handles.

Handles can be created or destroyed by user action without affecting the underlying open file description. Some of the ways to create them include `fcntl(2)`, `dup(2)`, `fdopen(3S)`, `fileno(3S)` and `fork(2)` (which duplicates existing ones into new processes). They can be destroyed by at least `fclose(3S)` and `close(2)`, and by the `exec` functions (see `exec(2)`), which close some file descriptors and destroy streams.

A file descriptor that is never used in an operation and could affect the file offset (for example `read(2)`, `write(2)`, or `lseek(2)`) is not considered a handle in this discussion, but could give rise to one (as a consequence of `fdopen()`, `dup()`, or `fork()`, for example). This exception does include the file

descriptor underlying a stream, whether created with **fopen()** or **fdopen()**, as long as it is not used directly by the application to affect the file offset. (The **read()** and **write()** functions implicitly affect the file offset; **lseek()** explicitly affects it.)

If two or more handles are used, and any one of them is a stream, their actions shall be coordinated as described below. If this is not done, the result is undefined.

A handle that is a stream is considered to be closed when either an **fclose()** or **freopen(3S)** is executed on it (the result of **freopen()** is a new stream for this discussion, which cannot be a handle on the same open file description as its previous value) or when the process owning that stream terminates the **exit(2)** or **abort(2)**. A file descriptor is closed by **close()**, **_exit()** (see **exit(2)**), or by one of the **exec** functions when **FD_CLOEXEC** is set on that file descriptor.

For a handle to become the active handle, the actions below must be performed between the last other user of the first handle (the current active handle) and the first other user of the second handle (the future active handle). The second handle then becomes the active handle. All activity by the application affecting the file offset on the first handle shall be suspended until it again becomes the active handle. (If a stream function has as an underlying function that affects the file offset, the stream function will be considered to affect the file offset. The underlying functions are described below.)

The handles need not be in the same process for these rules to apply. Note that after a **fork()**, two handles exist where one existed before. The application shall assure that, if both handles will ever be accessed, that they will both be in a state where the other could become the active handle first. The application shall prepare for a **fork()** exactly as if it were a change of active handle. (If the only action performed by one of the processes is one of the **exec** functions or **_exit()**, the handle is never accessed in that process.)

1. For the first handle, the first applicable condition below shall apply. After the actions required below are taken, the handle may be closed if it is still open.
 - a. If it is a file descriptor, no action is required.
 - b. If the only further action to be performed on any handle to this open file description is to close it, no action need be taken.
 - c. If it is a stream that is unbuffered, no action need be taken.
 - d. If it is a stream that is line-buffered and the last character written to the stream was a newline (that is, as if a `putc('\n')` was the most recent operation on that stream), no action need be taken.

- e. If it is a stream that is open for writing or append (but not also open for reading), either an `fflush(3S)` shall occur or the stream shall be closed.
 - f. If the stream is open for reading and it is at the end of the file (`feof(3S)` is true), no action need be taken.
 - g. If the stream is open with a mode that allows reading and the underlying open file description refers to a device that is capable of seeking, either an `fflush()` shall occur or the stream shall be closed.
 - h. Otherwise, the result is undefined.
2. For the second handle: if any previous active handle has called a function that explicitly changed the file offset, except as required above for the first handle, the application shall perform an `lseek()` or an `fseek(2)` (as appropriate to the type of the handle) to an appropriate location.
 3. If the active handle ceases to be accessible before the requirements on the first handle above have been met, the state of the open file description becomes undefined. This might occur, for example, during a `fork()` or an `_exit()`.
 4. The `exec` functions shall be considered to make inaccessible all streams that are open at the time they are called, independent of what streams or file descriptors may be available to the new process image.
 5. Implementation shall assure that an application, even one consisting of several processes, shall yield correct results (no data is lost or duplicated when writing, all data is written in order, except as requested by seeks) when the rules above are followed, regardless of the sequence of handles used. If the rules above are not followed, the result is unspecified. When these rules are followed, it is implementation defined whether, and under what conditions, all input is seen exactly once.

Use of stdio in Multithreaded Applications

All the `stdio` functions are safe unless they have the `_unlocked` suffix. Each `FILE` pointer has its own lock to guarantee that only one thread can access it. In the case that output needs to be synchronized, the lock for the `FILE` pointer can be acquired before performing a series of `stdio` operations. For example:

```
FILE iop;
:
:
flockfile(iop);
fprintf(iop, "hello ");
fprintf(iop, "world");
fputc(iop, 'a');
funlockfile(iop);
```

will print everything out together, blocking other threads that might want to write to the same file between calls to `fprintf()`.

An unlocked interface is available in case performance is an issue. For example:

```
flockfile(iop);
while (!feof(iop)) {
    *c++ = getc_unlocked(iop);
}
funlockfile(iop);
```

RETURN VALUES

Invalid stream pointers usually cause grave disorder, possibly including program termination. Individual function descriptions describe the possible error conditions.

SEE ALSO

`close(2)`, `lseek(2)`, `open(2)`, `pipe(2)`, `read(2)`, `write(2)`, `ctermid(3S)`, `cuserid(3S)`, `fclose(3S)`, `ferror(3S)`, `fopen(3S)`, `fread(3S)`, `fseek(3S)`, `flockfile(3S)`, `getc(3S)`, `gets(3S)`, `popen(3S)`, `printf(3S)`, `putc(3S)`, `puts(3S)`, `scanf(3S)`, `setbuf(3S)`, `system(3S)`, `tmpfile(3S)`, `tmpnam(3S)`, `ungetc(3S)`

| | |
|----------------------|---|
| NAME | str2sig, sig2str – translation between signal name and signal number |
| SYNOPSIS | <pre>#include <signal.h> int str2sig(const char * str, int *sigum); int sig2str(int sigum, char * str);</pre> |
| DESCRIPTION | <p>The <code>str2sig()</code> function translates the signal name <code>str</code> to a signal number, and stores that result in the location referenced by <code>sigum</code>. The name in <code>str</code> can be either the symbol for that signal, without the "SIG" prefix, or a decimal number. All the signal symbols defined in <code><sys/signal.h></code> are recognized. This means that both "CLD" and "CHLD" are recognized and return the same signal number, as do both "POLL" and "IO". For access to the signals in the range <code>SIGRTMIN</code> to <code>SIGRTMAX</code>, the first four signals match the strings "RTMIN", "RTMIN+1", "RTMIN+2", and "RTMIN+3" and the last four match the strings "RTMAX-3", "RTMAX-2", "RTMAX-1", and "RTMAX".</p> <p>The <code>sig2str()</code> function translates the signal number <code>sigum</code> to the symbol for that signal, without the "SIG" prefix, and stores that symbol at the location specified by <code>str</code>. The storage referenced by <code>str</code> should be large enough to hold the symbol and a terminating null byte. The symbol <code>SIG2STR_MAX</code> defined by <code><signal.h></code> gives the maximum size in bytes required.</p> |
| RETURN VALUES | <p>The <code>str2sig()</code> function returns 0 if it recognizes the signal name specified in <code>str</code>; otherwise, it returns -1.</p> <p>The <code>sig2str()</code> function returns 0 if the value <code>sigum</code> corresponds to a valid signal number; otherwise, it returns -1.</p> |
| EXAMPLES | <p>EXAMPLE 1 A sample program of using <code>str2sig()</code> function.</p> <pre style="border: 1px solid black; padding: 5px;">int i; char buf[STR2SIG_MAX]; /*storage for symbol */ str2sig("KILL",&i); /*stores 9 in i */ str2sig("9",&i); /* stores 9 in i */ sig2str(SIGKILL,buf); /* stores "KILL" in buf */ sig2str(9,buf); /* stores "KILL" in buf */</pre> |
| SEE ALSO | <code>kill(1)</code> , <code>strsignal(3C)</code> |

| | |
|--------------------|---|
| NAME | strncpy, streadd, strcadd, strecpy – copy strings, compressing or expanding escape codes |
| SYNOPSIS | <pre>cc [flag ...] file ... -lgen [library ...] #include <libgen.h> char * strncpy(char * <i>output</i>, const char * <i>input</i>); char * strcadd(char * <i>output</i>, const char * <i>input</i>); char * strecpy(char * <i>output</i>, const char * <i>input</i>, const char * <i>exceptions</i>); char * streadd(char * <i>output</i>, const char * <i>input</i>, const char * <i>exceptions</i>);</pre> |
| DESCRIPTION | <p>strncpy() copies the <i>input</i> string, up to a null byte, to the <i>output</i> string, compressing the C-language escape sequences (for example, \ , \\001)to the equivalent character. A null byte is appended to the output. The <i>output</i> argument must point to a space big enough to accommodate the result. If it is as big as the space pointed to by <i>input</i> it is guaranteed to be big enough. strncpy() returns the <i>output</i> argument.</p> <p>strcadd() is identical to strncpy() , except that it returns the pointer to the null byte that terminates the output.</p> <p>strecpy() copies the <i>input</i> string, up to a null byte, to the <i>output</i> string, expanding non-graphic characters to their equivalent C-language escape sequences (for example, \ , \\001). The <i>output</i> argument must point to a space big enough to accommodate the result; four times the space pointed to by <i>input</i> is guaranteed to be big enough (each character could become \\ and 3 digits). Characters in the <i>exceptions</i> string are not expanded. The <i>exceptions</i> argument may be zero, meaning all non-graphic characters are expanded. strecpy() returns the <i>output</i> argument.</p> <p>streadd() is identical to strecpy() , except that it returns the pointer to the null byte that terminates the output.</p> |
| EXAMPLES | <p>EXAMPLE 1 Example of expanding and compressing escape codes.</p> <pre>/* expand all but newline and tab */ strecpy(output, input, "\ \\t");</pre> |

```
/* concatenate and compress several strings */
cp = strcadd( output, input1 );
cp = strcadd( cp, input2 );
cp = strcadd( cp, input3 );
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

string(3C), **strfind(3G)**, **attributes(5)**

NOTES

When compiling multi-thread applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multi-thread applications.

| NAME | strcoll – string collation | | | | | | |
|----------------------|--|----------------|-----------------|----------|-------------------------|-----|---------|
| SYNOPSIS | <pre>#include <string.h> int strcoll(const char *s1, const char *s2);</pre> | | | | | | |
| DESCRIPTION | Both strcoll() and strxfrm(3C) provide for locale-specific string sorting. strcoll() is intended for applications in which the number of comparisons per string is small. When strings are to be compared a number of times, strxfrm(3C) is a more appropriate function because the transformation process occurs only once. | | | | | | |
| RETURN VALUES | <p>Upon successful completion, strcoll() returns an integer greater than, equal to, or less than zero in direct correlation to whether string <i>s1</i> is greater than, equal to, or less than the string <i>s2</i>. The comparison is based on strings interpreted as appropriate to the program's locale for category LC_COLLATE (see setlocale(3C)).</p> <p>On error, strcoll() may set <code>errno</code>, but no return value is reserved to indicate an error.</p> | | | | | | |
| ERRORS | <p>The strcoll() function may fail if:</p> <p>EINVAL The <i>s1</i> or <i>s2</i> arguments contain characters outside the domain of the collating sequence.</p> | | | | | | |
| FILES | <pre>/usr/lib/locale/locale/locale.so.* LC_COLLATE database for locale</pre> | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe with exceptions</td> </tr> <tr> <td>CSI</td> <td>Enabled</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe with exceptions | CSI | Enabled |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| MT-Level | MT-Safe with exceptions | | | | | | |
| CSI | Enabled | | | | | | |
| SEE ALSO | localedef(1) , setlocale(3C) , string(3C) , strxfrm(3C) , wsxfrm(3C) , attributes(5) , environ(5) | | | | | | |
| NOTES | The strcoll() function can be used safely in multithreaded applications, as long as setlocale(3C) is not being called to change the locale. | | | | | | |

NAME | strerror – get error message string

SYNOPSIS | #include <string.h>

char ***strerror**(int *errnum*);

DESCRIPTION | The **strerror()** function maps the error number in *errnum* to an error message string, and returns a pointer to that string. It uses the same set of error messages as **perror**(3C). The returned string should not be overwritten.

RETURN VALUES | The **strerror()** function returns `NULL` if *errnum* is out-of-range.

ATTRIBUTES | See **attributes**(5) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO | **gettext**(3C), **perror**(3C), **setlocale**(3C), **attributes**(5)

NOTES | If the application is linked with `-lintl`, then messages returned from this function are in the native language specified by the `LC_MESSAGES` locale category; see **setlocale**(3C).

| NAME | strfind, strrspn, strtrns, str – string manipulations | | | | |
|--------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lgen [library ...] #include <libgen.h> int strfind(const char * as1, const char * as2); char * strrspn(const char * string, const char * tc); char * strtrns(const char * string, const char * old, const char * new, char * result);</pre> | | | | |
| DESCRIPTION | <p>strfind() returns the offset of the first occurrence of the second string, <i>as2</i> , if it is a substring of string <i>as1</i> . If the second string is not a substring of the first string strfind() returns <i>-1</i> .</p> <p>strrspn() returns a pointer to the first character in the string that is not one of the characters in <i>tc</i> .</p> <p>strtrns() transforms <i>string</i> and copies it into <i>result</i> . Any character that appears in <i>old</i> is replaced with the character in the same position in <i>new</i> . The <i>new</i> result is returned.</p> | | | | |
| EXAMPLES | <p>EXAMPLE 1 An example of the strfind() function.</p> <pre style="border: 1px solid black; padding: 5px;">/* find offset to substring "hello" within as1 */ i = strfind(as1, "hello"); /* trim junk</pre> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | string(3C) , attributes(5) | | | | |
| NOTES | When compiling multi-thread applications, the _REENTRANT flag must be defined on the compile line. This flag should only be used in multi-thread applications. | | | | |

| | |
|--------------------|---|
| NAME | strfmon – convert monetary value to string |
| SYNOPSIS | <pre>#include <monetary.h> ssize_t strfmon(char *s, size_t maxsize, const char *format, ...);</pre> |
| DESCRIPTION | <p>The strfmon() function places characters into the array pointed to by <i>s</i> as controlled by the string pointed to by <i>format</i>. No more than <i>maxsize</i> bytes are placed into the array.</p> <p>The format is a character string that contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in the fetching of zero or more arguments which are converted and formatted. The results are undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments are simply ignored.</p> <p>A conversion specification consists of the following sequence:</p> <ul style="list-style-type: none"> ■ a % character ■ optional flags ■ optional field width ■ optional left precision ■ optional right precision ■ a required conversion character that determines the conversion to be performed. |
| Flags | <p>One or more of the following optional flags can be specified to control the conversion:</p> <p>=<i>f</i> An = followed by a single character <i>f</i> which is used as the numeric fill character. The fill character must be representable in a single byte in order to work with precision and width counts. The default numeric fill character is the space character. This flag does not affect field width filling which always uses the space character. This flag is ignored unless a left precision (see below) is specified.</p> <p>^ Do not format the currency amount with grouping characters. The default is to insert the grouping characters if defined for the current locale.</p> <p>+ or (Specify the style of representing positive and negative currency amounts. Only one of '+' or '(' may be specified. If '+' is specified, the locale's equivalent of + and '-' are used (for example, in the U.S.A.: the empty string if positive and '-' if negative). If '(' is specified,</p> |

| | | |
|------------------------------|-----------|--|
| | | negative amounts are enclosed within parentheses. If neither flag is specified, the '+' style is used. |
| | ! | Suppress the currency symbol from the output conversion. |
| | - | Specify the alignment. If this flag is present all fields are left-justified (padded to the right) rather than right-justified. |
| Field Width | w | A decimal digit string <i>w</i> specifying a minimum field width in bytes in which the result of the conversion is right-justified (or left-justified if the flag '-' is specified). The default is zero. |
| Left Precision | #n | <p>A '#' followed by a decimal digit string <i>n</i> specifying a maximum number of digits expected to be formatted to the left of the radix character. This option can be used to keep the formatted output from multiple calls to the strfmon() aligned in the same columns. It can also be used to fill unused positions with a special character as in \$***123.45. This option causes an amount to be formatted as if it has the number of digits specified by <i>n</i>. If more than <i>n</i> digit positions are required, this conversion specification is ignored. Digit positions in excess of those actually required are filled with the numeric fill character (see the =<i>f</i> flag above).</p> <p>If grouping has not been suppressed with the '^' flag, and it is defined for the current locale, grouping separators are inserted before the fill characters (if any) are added. Grouping separators are not applied to fill characters even if the fill character is a digit.</p> <p>To ensure alignment, any characters appearing before or after the number in the formatted output such as currency or sign symbols are padded as necessary with space characters to make their positive and negative formats an equal length.</p> |
| Right Precision | .p | A period followed by a decimal digit string <i>p</i> specifying the number of digits after the radix character. If the value of the right precision <i>p</i> is zero, no radix character appears. If a right precision is not included, a default specified by the current locale is used. The amount being formatted is rounded to the specified number of digits prior to formatting. |
| Conversion Characters | | The conversion characters and their meanings are: |

- i** The `double` argument is formatted according to the locale's international currency format (for example, in the U.S.A.: USD 1,234.56).
- n** The `double` argument is formatted according to the locale's national currency format (for example, in the U.S.A.: \$1,234.56).
- %** Convert to a %; no argument is converted. The entire conversion specification must be %%.

Locale Information

The `LC_MONETARY` category of the program's locale affects the behavior of this function including the monetary radix character (which may be different from the numeric radix character affected by the `LC_NUMERIC` category), the grouping separator, the currency symbols and formats. The international currency symbol should be in conformance with the ISO 4217: 1987 standard.

RETURN VALUES

If the total number of resulting bytes (including the terminating null byte) is not more than `maxsize`, `strfmon()` returns the number of bytes placed into the array pointed to by `s`, not including the terminating null byte. Otherwise, `-1` is returned, the contents of the array are indeterminate, and `errno` is set to indicate the error.

ERRORS

The `strfmon()` function will fail if:

ENOSYS The function is not supported.

E2BIG Conversion stopped due to lack of space in the buffer.

EXAMPLES

EXAMPLE 1 A sample output of `strfmon()`.

Given a locale for the U.S.A. and the values 123.45, `-123.45`, and `3456.781`:

| Conversion Specification | Output | Comments |
|--------------------------|-------------------------------------|--|
| <code>%n</code> | \$123.45 -\$123.45 \$3,456.78 | default formatting |
| <code>%11n</code> | \$123.45 -\$123.45 \$3,456.78 | right align within an 11 character field |

| | | |
|---------|--|---|
| %#5n | \$123.45 -\$123.45 \$3,456.78 | aligned columns for values up to 99,999 |
| %=#5n | \$***123.45 -\$***123.45 \$*3,456.78 | specify a fill character |
| %=0#5n | \$000123.45 -\$000123.45 \$03,456.78 | fill characters do not use grouping even if the fill character is a digit |
| %^#5n | \$123.45 -\$123.45 \$3456.78 | disable the grouping separator |
| %^#5.0n | \$123 -\$123 \$3457 | round off to whole units |
| %^#5.4n | \$123.4500 -\$123.4500 \$3456.7810 | increase the precision |
| %(#5n | 123.45 (\$123.45) \$3,456.78 | use an alternative pos/neg style |
| %!(#5n | 123.45 (123.45) 3,456.78 | disable the currency symbol |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |
| CSI | Enabled |

SEE ALSO `localeconv(3C)`, `setlocale(3C)`, `attributes(5)`

NOTES This function can be used safely in multithreaded applications, as long as `setlocale(3C)` is not called to change the locale.

| | |
|--------------------|--|
| NAME | strptime, cftime, ascftime – convert date and time to string |
| SYNOPSIS | <pre>#include <time.h> size_t strptime(const char * s, size_t <i>maxsize</i>, const char * <i>format</i>, const struct tm * <i>timeptr</i>); int cftime(char * s, char * <i>format</i>, const time_t * <i>clock</i>); int ascftime(char * s, const char * <i>format</i>, const struct tm * <i>timeptr</i>);</pre> |
| DESCRIPTION | <p>The strptime(), ascftime(), and cftime() functions place bytes into the array pointed to by <i>s</i> as controlled by the string pointed to by <i>format</i>. The <i>format</i> string consists of zero or more conversion specifications and ordinary characters. A conversion specification consists of a '%' (percent) character and one or two terminating conversion characters that determine the conversion specification's behavior. All ordinary characters (including the terminating null byte) are copied unchanged into the array pointed to by <i>s</i>. If copying takes place between objects that overlap, the behavior is undefined. For strptime(), no more than <i>maxsize</i> bytes are placed into the array.</p> <p>If <i>format</i> is (char *)0, then the locale's default format is used. For strptime() the default format is the same as %c; for cftime() and ascftime() the default format is the same as %C. cftime() and ascftime() first try to use the value of the environment variable CFTIME, and if that is undefined or empty, the default format is used.</p> <p>Each conversion specification is replaced by appropriate characters as described in the following list. The appropriate characters are determined by the LC_TIME category of the program's locale and by the values contained in the structure pointed to by <i>timeptr</i> for strptime() and ascftime(), and by the time represented by <i>clock</i> for cftime().</p> <ul style="list-style-type: none"> %% Same as %. %a Locale's abbreviated weekday name. %A Locale's full weekday name. %b Locale's abbreviated month name. %B Locale's full month name. %c Locale's appropriate date and time representation. <p>Default</p> <ul style="list-style-type: none"> %C Locale's date and time representation as produced by date(1). |

Standard-conforming

| | |
|----|---|
| %C | Century number (the year divided by 100 and truncated to an integer as a decimal number [1,99]); single digits are preceded by 0; see standards(5) . |
| %d | Day of month [1,31]; single digits are preceded by 0. |
| %D | Date as %m / %d / %Y . |
| %e | Day of month [1,31]; single digits are preceded by a space. |
| %g | Week-based year within century [00,99]. |
| %G | Week-based year, including the century [0000,9999]. |
| %h | Locale's abbreviated month name. |
| %H | Hour (24-hour clock) [0,23]; single digits are preceded by 0. |
| %I | Hour (12-hour clock) [1,12]; single digits are preceded by 0. |
| %j | Day number of year [1,366]; single digits are preceded by 0. |
| %k | Hour (24-hour clock) [0,23]; single digits are preceded by a blank. |
| %l | Hour (12-hour clock) [1,12]; single digits are preceded by a blank. |
| %m | Month number [1,12]; single digits are preceded by 0. |
| %M | Minute [00,59]; leading 0 is permitted but not required. |
| %n | Insert a NEWLINE. |
| %p | Locale's equivalent of either a.m. or p.m. |
| %r | Appropriate time representation in 12-hour clock format with %p . |
| %R | Time as %H : %M . |
| %S | Seconds [00,61]; the range of values is [00,61] rather than [00,59] to allow for the occasional leap second and even more occasional double leap second. |
| %t | Insert a TAB. |
| %T | Time as %H : %M : %S . |
| %u | Weekday as a decimal number [1,7], with 1 representing Sunday. |

- `%U` Week number of year as a decimal number [00,53], with Sunday as the first day of week 1.
- `%V` The ISO 8601 week number as a decimal number [01,53]. In the ISO 8601 week-based system, weeks begin on a Monday and week 1 of the year is the week that includes both January 4th and the first Thursday of the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year. See NOTES below.
- `%w` Weekday as a decimal number [0,6], with 0 representing Sunday.
- `%W` Week number of year as a decimal number [00,53], with Monday as the first day of week 1.
- `%x` Locale's appropriate date representation.
- `%X` Locale's appropriate time representation.
- `%y` Year within century [00,99].
- `%Y` Year, including the century (for example 1993).
- `%Z` Time zone name or abbreviation, or no bytes if no time zone information exists.

If a conversion specification does not correspond to any of the above or to any of the modified conversion specifications listed below, the behavior is undefined and 0 is returned.

The difference between `%U` and `%W` (and also between modified conversion specifications `%OU` and `%OW`) lies in which day is counted as the first of the week. Week number 1 is the first week in January starting with a Sunday for `%U` or a Monday for `%W`. Week number 0 contains those days before the first Sunday or Monday in January for `%U` and `%W`, respectively.

Modified Conversion Specifications

Some conversion specifications can be modified by the `E` and `O` modifiers to indicate that an alternate format or specification should be used rather than the one normally used by the unmodified conversion specification. If the alternate format or specification does not exist in the current locale, the behavior will be as if the unmodified specification were used.

- `%Ec` Locale's alternate appropriate date and time representation.
- `%EC` Name of the base year (period) in the locale's alternate representation.
- `%Eg` Offset from `%EC` of the week-based year in the locale's alternative representation.

| | |
|-----|---|
| %EG | Full alternative representation of the week-based year. |
| %Ex | Locale's alternate date representation. |
| %EX | Locale's alternate time representation. |
| %Ey | Offset from %EC (year only) in the locale's alternate representation. |
| %EY | Full alternate year representation. |
| %Od | Day of the month using the locale's alternate numeric symbols. |
| %Oe | Same as %Od . |
| %Og | Week-based year (offset from %C)in the locale's alternate representation and using the locale's alternate numeric symbols. |
| %OH | Hour (24-hour clock) using the locale's alternate numeric symbols. |
| %OI | Hour (12-hour clock) using the locale's alternate numeric symbols. |
| %Om | Month using the locale's alternate numeric symbols. |
| %OM | Minutes using the locale's alternate numeric symbols. |
| %OS | Seconds using the locale's alternate numeric symbols. |
| %Ou | Weekday as a number in the locale's alternate numeric symbols. |
| %OU | Week number of the year (Sunday as the first day of the week) using the locale's alternate numeric symbols. |
| %Ow | Number of the weekday (Sunday=0) using the locale's alternate numeric symbols. |
| %OW | Week number of the year (Monday as the first day of the week) using the locale's alternate numeric symbols. |
| %Oy | Year (offset from %C)in the locale's alternate representation and using the locale's alternate numeric symbols. |

Selecting the Output Language

By default, the output of **strftime()** , **cftime()** , and **ascftime()** appear in U.S. English. The user can request that the output of **strftime()** , **cftime()** , or **ascftime()** be in a specific language by setting the LC_TIME category using **setlocale()** .

Time Zone

Local time zone information is used as though `tzset(3C)` were called.

RETURN VALUES

The `strptime()`, `ctime()`, and `asctime()` functions return the number of characters placed into the array pointed to by `s`, not including the terminating null character. If the total number of resulting characters including the terminating null character is more than `maxsize`, `strptime()` returns 0 and the contents of the array are indeterminate.

EXAMPLES

EXAMPLE 1 An example of the `strptime()` function.

The following example illustrates the use of `strptime()` for the POSIX locale. It shows what the string in `str` would look like if the structure pointed to by `tm_ptr` contains the values corresponding to Thursday, August 28, 1986 at 12:44:36.

```
strptime (str, strsize, "%A %b %d %j", tm_ptr)
```

This results in `str` containing "Thursday Aug 28 240".

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |
| CSI | Enabled |

SEE ALSO

`date(1)`, `ctime(3C)`, `mktime(3C)`, `setlocale(3C)`, `strptime(3C)`, `tzset(3C)`, `TIMEZONE(4)`, `attributes(5)`, `environ(5)`, `standards(5)`

NOTES

The conversion specification for `%V` was changed in the Solaris 7 release. This change was based on the public review draft of the ISO C9x standard at that time. Previously, the specification stated that if the week containing 1 January had fewer than four days in the new year, it became week 53 of the previous year. The ISO C9x standard committee subsequently recongnized that that specification had been incorrect.

The conversion specifications for `%g`, `%G`, `%Eg`, `%EG`, and `%Og` were added in the Solaris 7 release. This change was based on the public review draft of the ISO C9x standard at that time. These specifications are evolving. If the ISO C9x standard is finalized with a different conclusion, these specifications will change to conform to the ISO C9x standard decision.

| | |
|---|---|
| NAME | string, strcasecmp, strncasecmp, strcat, strncat, strchr, strchr, strcmp, strncmp, strcpy, strncpy, strcspn, strspn, strdup, strlen, strpbrk, strstr, strtok, strtok_r - string operations |
| SYNOPSIS | <pre>#include <strings.h> int strcasecmp(const char * s1, const char * s2); int strncasecmp(const char * s1, const char * s2, size_t n); #include <string.h> char * strcat(char * s1, const char * s2); char * strncat(char * s1, const char * s2, size_t n); char * strchr(const char * s, int c); char * strrchr(const char * s, int c); int strcmp(const char * s1, const char * s2); int strncmp(const char * s1, const char * s2, size_t n); char * strcpy(char * s1, const char * s2); char * strncpy(char * s1, const char * s2, size_t n); size_t strcspn(const char * s1, const char * s2); size_t strspn(const char * s1, const char * s2); char * strdup(const char * s1); size_t strlen(const char * s); char * strpbrk(const char * s1, const char * s2); char * strstr(const char * s1, const char * s2); char * strtok(char * s1, const char * s2); char * strtok_r(char * s1, const char * s2, char ** lasts);</pre> |
| DESCRIPTION | <p>The arguments <i>s</i>, <i>s1</i>, and <i>s2</i> point to strings (arrays of characters terminated by a null character). The strcat(), strncat(), strcpy(), strncpy(), strtok(), and strtok_r() functions all alter their first argument. These functions do not check for overflow of the array pointed to by the first argument.</p> <p>The strcasecmp() and strncasecmp() functions are case-insensitive versions of strcmp() and strncmp() respectively, described below. They assume the ASCII</p> |
| strcasecmp() , strncasecmp() | |

| | |
|------------------------------------|--|
| | character set and ignore differences in case when comparing lower and upper case characters. |
| strcat() , strncat() | The strcat() function appends a copy of string <i>s2</i> , including the terminating null character, to the end of string <i>s1</i> . The strncat() function appends at most <i>n</i> characters. Each returns a pointer to the null-terminated result. The initial character of <i>s2</i> overrides the null character at the end of <i>s1</i> . |
| strchr() , strrchr() | The strchr() function returns a pointer to the first occurrence of <i>c</i> (converted to a <code>char</code>)in string <i>s</i> , or a null pointer if <i>c</i> does not occur in the string. The strrchr() function returns a pointer to the last occurrence of <i>c</i> . The null character terminating a string is considered to be part of the string. |
| strcmp() , strncmp() | The strcmp() function compares two strings byte-by-byte, according to the ordering of your machine's character set. The function returns an integer greater than, equal to, or less than 0, if the string pointed to by <i>s1</i> is greater than, equal to, or less than the string pointed to by <i>s2</i> respectively. The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of bytes that differ in the strings being compared. The strncmp() function makes the same comparison but looks at a maximum of <i>n</i> bytes. Bytes following a null byte are not compared. |
| strcpy() , strncpy() | The strcpy() function copies string <i>s2</i> to <i>s1</i> , including the terminating null character, stopping after the null character has been copied. The strncpy() function copies exactly <i>n</i> bytes, truncating <i>s2</i> or adding null characters to <i>s1</i> if necessary. The result will not be null-terminated if the length of <i>s2</i> is <i>n</i> or more. Each function returns <i>s1</i> . |
| strcspn() , strspn() | The strcspn() function returns the length of the initial segment of string <i>s1</i> that consists entirely of characters not from string <i>s2</i> . The strspn() function returns the length of the initial segment of string <i>s1</i> that consists entirely of characters from string <i>s2</i> . |
| strdup() | The strdup() function returns a pointer to a new string that is a duplicate of the string pointed to by <i>s1</i> . The space for the new string is obtained using <code>malloc(3C)</code> . If the new string cannot be created, a null pointer is returned. |
| strlen() | The strlen() function returns the number of bytes in <i>s</i> , not including the terminating null character. |
| strpbrk() | The strpbrk() function returns a pointer to the first occurrence in string <i>s1</i> of any character from string <i>s2</i> , or a null pointer if no character from <i>s2</i> exists in <i>s1</i> . |

- strstr(), strstr()** The **strstr()** function locates the first occurrence of the string *s2* (excluding the terminating null character) in string *s1*. The **strstr()** function returns a pointer to the located string, or a null pointer if the string is not found. If *s2* points to a string with zero length (that is, the string ""), the function returns *s1*.
- strtok()** The **strtok()** function can be used to break the string pointed to by *s1* into a sequence of tokens, each of which is delimited by one or more characters from the string pointed to by *s2*. The **strtok()** function considers the string *s1* to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string *s2*. The first call (with pointer *s1* specified) returns a pointer to the first character of the first token, and will have written a null character into *s1* immediately following the returned token. The function keeps track of its position in the string between separate calls, so that subsequent calls (which must be made with the first argument being a null pointer) will work through the string *s1* immediately following that token. In this way subsequent calls will work through the string *s1* until no tokens remain. The separator string *s2* may be different from call to call. When no token remains in *s1*, a null pointer is returned.
- strtok_r()** The **strtok_r()** function has the same functionality as **strtok()** except that a pointer to a string placeholder *lasts* must be supplied by the caller. The *lasts* pointer is to keep track of the next substring in which to search for the next token.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|------------------|
| MT-Level | See NOTES below. |

SEE ALSO

malloc(3C), **setlocale(3C)**, **strxfrm(3C)**, **attributes(5)**

NOTES

The **strtok_r()** function is as proposed in the POSIX.4a Draft #6 document, and is subject to change to be compliant to the standard when it is accepted.

When compiling multithreaded applications, the **_REENTRANT** flag must be defined on the compile line. This flag should only be used in multithreaded applications.

All of these functions assume the default locale "C." For some locales, **strxfrm()** should be applied to the strings before they are passed to the functions.

The **strcasemp()**, **strcat()**, **strchr()**, **strcmp()**, **strcpy()**, **strcspn()**, **strdup()**, **strlen()**, **strncasemp()**, **strncat()**, **strncmp()**, **strncpy()**,

`strpbrk()`, `strchr()`, `strspn()`, and `strstr()` functions are MT-Safe in multithreaded applications.

The `strtok()` function is Unsafe in multithreaded applications. The `strtok_r()` function should be used instead.

| | |
|--------------------|---|
| NAME | string_to_decimal, file_to_decimal, func_to_decimal – parse characters into decimal record |
| SYNOPSIS | <pre>#include <floatingpoint.h> void string_to_decimal(char ** pc, int nmax, int fortran_conventions, decimal_record * pd, enum decimal_string_form * pform, char ** pechar); void func_to_decimal(char ** pc, int nmax, int fortran_conventions, decimal_record * pd, enum decimal_string_form * pform, char ** pechar, int (* pget)(void), int * pntread, int (* punget)(int c)); #include <stdio.h> void file_to_decimal(char ** pc, int nmax, int fortran_conventions, decimal_record * pd, enum decimal_string_form * pform, char ** pechar, FILE * pf, int * pntread);</pre> |
| DESCRIPTION | <p>The <code>char_to_decimal</code> functions parse a numeric token from at most <code>nmax</code> characters in a string <code>**pc</code> or file <code>*pf</code> or function <code>(*pget)()</code> into a decimal record <code>*pd</code>, classifying the form of the string in <code>*pform</code> and <code>*pechar</code>. The accepted syntax is intended to be sufficiently flexible to accommodate many languages: <i>whitespace value</i> or <i>whitespace sign value</i>, where <i>whitespace</i> is any number of characters defined by <i>isspace</i> in <code><ctype.h></code>, <i>sign</i> is either of <code>[+-]</code>, and <i>value</i> can be <i>number</i>, <i>nan</i>, or <i>inf</i>. <i>inf</i> can be <code>INF</code> (<i>inf_form</i>) or <code>INFINITY</code> (<i>infinity_form</i>) without regard to case. <i>nan</i> can be <code>NAN</code> (<i>nan_form</i>) or <code>NAN(<i>nstring</i>)</code> (<i>nanstring_form</i>) without regard to case; <i>nstring</i> is any string of characters not containing <code>'</code> or <code>NULL</code>; <i>nstring</i> is copied to <code>pd->ds</code> and, currently, not used subsequently. <i>number</i> consists of <i>significand</i> or <i>significand efield</i> where <i>significand</i> must contain one or more digits and may contain one point; possible forms are</p> <pre> <i>digits</i> (int_form) <i>digits.</i> (intdot_form) .<i>digits</i> (dotfrac_form) <i>digits.digits</i> (intdotfrac_form)</pre> <p><i>efield</i> consists of <i>echar digits</i> or <i>echar sign digits</i>, where <i>echar</i> is one of <code>[Ee]</code>, and <i>digits</i> contains one or more digits.</p> |

When *fortran_conventions* is nonzero, additional input forms are accepted according to various Fortran conventions:

- 0** no Fortran conventions
- 1** Fortran list-directed input conventions
- 2** Fortran formatted input conventions, ignore blanks (BN)
- 3** Fortran formatted input conventions, blanks are zeros (BZ)

When *fortran_conventions* is nonzero, *echar* may also be one of [DdQq], and *efield* may also have the form

sign digits .

When *fortran_conventions* ≥ 2 , blanks may appear in the *digits* strings for the integer, fraction, and exponent fields and may appear between *echar* and the exponent sign and after the infinity and NaN forms. If *fortran_conventions* $== 2$, the blanks are ignored. When *fortran_conventions* $== 3$, the blanks that appear in *digits* strings are interpreted as zeros, and other blanks are ignored.

When *fortran_conventions* is zero, the current locale's decimal point character is used as the decimal point; when *fortran_conventions* is nonzero, the period is used as the decimal point.

The form of the accepted decimal string is placed in **pform* . If an *efield* is recognized, **pechar* is set to point to the *echar* .

On input, **pc* points to the beginning of a character string buffer of length $\geq nmax$. On output, **pc* points to a character in that buffer, one past the last accepted character. **string_to_decimal()** gets its characters from the buffer; **file_to_decimal()** gets its characters from **pf* and records them in the buffer, and places a null after the last character read. **func_to_decimal()** gets its characters from an int function (***pget()**) .

The scan continues until no more characters could possibly fit the acceptable syntax or until *nmax* characters have been scanned. If the *nmax* limit is not reached then at least one extra character will usually be scanned that is not part of the accepted syntax. **file_to_decimal()** and **func_to_decimal()** set **pnread* to the number of characters read from the file; if greater than *nmax* , some characters were lost. If no characters were lost, **file_to_decimal()** and **func_to_decimal()** attempt to push back, with **ungetc(3S)** or (***punget()**) , as many as possible of the excess characters read, adjusting **pnread* accordingly. If all unget calls are successful, then ***pc* will be NULL . No push back will be attempted if (***punget()**) is NULL .

Typical declarations for ** pget()* and ** punget()* are:

```

int xget(void)
\011{ . . . }
\011int (*pget)(void) = xget;
\011int xunget(int c)
\011{ . . . }
int (*punget)(int) = xunget;

```

If no valid number was detected, *pd* \rightarrow *fpclass* is set to *fp_signaling*, *pc* is unchanged, and *pform* is set to *invalid_form*.

atof(3C) and **strtod(3C)** use **string_to_decimal()**. **scanf(3S)** uses **file_to_decimal()**.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

ctype(3C), **localeconv(3C)**, **scanf(3S)**, **setlocale(3C)**, **strtod(3C)**, **ungetc(3S)**, **attributes(5)**

| | |
|----------------------------------|--|
| NAME | strptime – date and time conversion |
| SYNOPSIS | <pre>#include <time.h> char *strptime(const char *buf, const char *format, struct tm *tm);</pre> |
| DESCRIPTION | <p>The strptime() function converts the character string pointed to by <i>buf</i> to values which are stored in the <code>tm</code> structure pointed to by <i>tm</i>, using the format specified by <i>format</i>.</p> <p>The <i>format</i> argument is composed of zero or more conversion specifications. Each conversion specification is composed of a “%” (percent) character followed by one or two conversion characters which specify the replacement required. One or more white space characters (as specified by <code>isspace(3C)</code>) may precede or follow a conversion specification. There must be white-space or other non-alphanumeric characters between any two conversion specifications.</p> |
| Conversion Specifications | <p>The following conversion specifications are supported:</p> <ul style="list-style-type: none"> %% Same as %. %a Day of week, using the locale’s weekday names; either the abbreviated or full name may be specified. %A Same as %a. %b Month, using the locale’s month names; either the abbreviated or full name may be specified. %B Same as %b. %c Locale’s appropriate date and time representation. %C Century number [0,99]; leading zero is permitted but not required. %d Day of month [1,31]; leading zero is permitted but not required. %D Date as %m/%d/%y. %e Same as %d. %h Same as %b. %H Hour (24-hour clock) [0,23]; leading zero is permitted but not required. %I Hour (12-hour clock) [1,12]; leading zero is permitted but not required. %j Day number of the year [1,366]; leading zeros are permitted but not required. |

| | |
|-----------------|--|
| <code>%m</code> | Month number [1,12]; leading zero is permitted but not required. |
| <code>%M</code> | Minute [0-59]; leading zero is permitted but not required. |
| <code>%n</code> | Any white space. |
| <code>%p</code> | Locale's equivalent of either a.m. or p.m. |
| <code>%r</code> | Appropriate time representation in the 12-hour clock format with <code>%p</code> . |
| <code>%R</code> | Time as <code>%H:%M</code> . |
| <code>%S</code> | Seconds [0,61]; leading zero is permitted but not required. The range of values is [00,61] rather than [00,59] to allow for the occasional leap second and even more occasional double leap second. |
| <code>%t</code> | Any white space. |
| <code>%T</code> | Time as <code>%H:%M:%S</code> . |
| <code>%U</code> | Week number of the year as a decimal number [0,53], with Sunday as the first day of the week; leading zeros are permitted but not required. |
| <code>%w</code> | Weekday as a decimal number [0,6], with 0 representing Sunday. |
| <code>%W</code> | Week number of the year as a decimal number [0,53], with Monday as the first day of the week; leading zero is permitted but not required. |
| <code>%x</code> | Locale's appropriate date representation. |
| <code>%X</code> | Locale's appropriate time representation. |
| <code>%y</code> | The year within century. When a century is not otherwise specified, values in the range 69-99 refer to years in the twentieth century (1969 to 1999 inclusive); values in the range 00-68 refer to years in the twenty-first century (2000 to 2068 inclusive). Leading zeros are permitted but not required. |
| <code>%Y</code> | Year, including the century (for example, 1993). |
| <code>%Z</code> | Timezone name or no characters if no time zone information exists. Local timezone information is used as though <code>strptime()</code> called <code>tzset()</code> (see <code>ctime(3C)</code>). Errors may not be detected. This behavior is subject to change in a future release. |

Modified Conversion Specifications

Some conversion specifications can be modified by the `E` and `O` modifier characters to indicate that an alternate format or specification should be used rather than the one normally used by the unmodified specification. If the alternate format or specification does not exist in the current locale, the behavior will be as if the unmodified conversion specification were used.

- `%Ec` Locale's alternate appropriate date and time representation.
- `%EC` Name of the base year (era) in the locale's alternate representation.
- `%Ex` Locale's alternate date representation.
- `%EX` Locale's alternate time representation.
- `%Ey` Offset from `%EC` (year only) in the locale's alternate representation.
- `%EY` Full alternate year representation.
- `%Od` Day of the month using the locale's alternate numeric symbols.
- `%Oe` Same as `%Od`.
- `%OH` Hour (24-hour clock) using the locale's alternate numeric symbols.
- `%OI` Hour (12-hour clock) using the locale's alternate numeric symbols.
- `%Om` Month using the locale's alternate numeric symbols.
- `%OM` Minutes using the locale's alternate numeric symbols.
- `%OS` Seconds using the locale's alternate numeric symbols.
- `%OU` Week number of the year (Sunday as the first day of the week) using the locale's alternate numeric symbols.
- `%Ow` Number of the weekday (Sunday=0) using the locale's alternate numeric symbols.
- `%OW` Week number of the year (Monday as the first day of the week) using the locale's alternate numeric symbols.
- `%Oy` Year (offset from `%C`) in the locale's alternate representation and using the locale's alternate numeric symbols.

General Specifications

A conversion specification that is an ordinary character is executed by scanning the next character from the buffer. If the character scanned from the

buffer differs from the one comprising the specification, the specification fails, and the differing and subsequent characters remain unscanned.

A series of specifications composed of %n, %t, white-space characters or any combination is executed by scanning up to the first character that is not white space (which remains unscanned), or until no more characters can be scanned. White space is defined by `isspace(3C)`.

Any other conversion specification is executed by scanning characters until a character matching the next specification is scanned, or until no more characters can be scanned. These characters, except the one matching the next specification, are then compared to the locale values associated with the conversion specifier. If a match is found, values for the appropriate *tm* structure members are set to values corresponding to the locale information. If no match is found, `strptime()` fails and no more characters are scanned.

The month names, weekday names, era names, and alternate numeric symbols can consist of any combination of upper and lower case letters. The user can request that the input date or time specification be in a specific language by setting the LC_TIME category using `setlocale(3C)`.

RETURN VALUES

Upon successful completion, `strptime()` returns a pointer to the character following the last character parsed. Otherwise, a null pointer is returned.

USAGE

Several “same as” formats, and the special processing of white-space characters are provided in order to ease the use of identical *format* strings for `strptime(3C)` and `strptime()`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |
| CSI | Enabled |

SEE ALSO

`ctime(3C)`, `getdate(3C)`, `isspace(3C)`, `setlocale(3C)`, `strptime(3C)`, `attributes(5)`, `environ(5)`

| NAME | strsignal – get name of signal | | | | |
|----------------------|--|----------------|-----------------|----------|------|
| SYNOPSIS | #include <string.h> char *strsignal(int sig); | | | | |
| DESCRIPTION | The strsignal() function maps the signal number in <i>sig</i> to a string describing the signal and returns a pointer to that string. It uses the same set of the messages as psignal(3C) . The returned string should not be overwritten. | | | | |
| RETURN VALUES | The strsignal() function returns <code>NULL</code> if <i>sig</i> is not a valid signal number. | | | | |
| USAGE | If the application is linked with <code>-lintl</code> , messages returned from this function are in the native language specified by the <code>LC_MESSAGES</code> locale category; see setlocale(3C) . | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Safe | | | | |
| SEE ALSO | gettext(3C) , psignal(3C) , setlocale(3C) , str2sig(3C) , attributes(5) | | | | |

| | |
|--------------------|--|
| NAME | strtod, atof – convert string to double-precision number |
| SYNOPSIS | <pre>#include <stdlib.h> double strtod(const char * str, char ** endptr); double atof(const char * str);</pre> |
| DESCRIPTION | <p>The strtod() function converts the initial portion of the string pointed to by <i>str</i> to type <code>double</code> representation. First it decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by <code>isspace(3C)</code>); a subject sequence interpreted as a floating-point constant; and a final string of one or more unrecognized characters, including the terminating null byte of the input string. Then it attempts to convert the subject sequence to a floating-point number, and returns the result.</p> <p>The expected form of the subject sequence is an optional + or – sign, then a non-empty sequence of digits optionally containing a radix character, then an optional exponent part. An exponent part consists of e or E, followed by an optional sign, followed by one or more decimal digits. The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence is empty if the input string is empty or consists entirely of white-space characters, or if the first character that is not white space is other than a sign, a digit or a radix character.</p> <p>If the subject sequence has the expected form, the sequence starting with the first digit or the radix character (whichever occurs first) is interpreted as a floating constant of the C language, except that the radix character is used in place of a period, and that if neither an exponent part nor a radix character appears, a radix character is assumed to follow the last digit in the string. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in the object pointed to by <i>endptr</i>, provided that <i>endptr</i> is not a null pointer.</p> <p>The radix character is defined in the program's locale (category <code>LC_NUMERIC</code>). In the POSIX locale, or in a locale where the radix character is not defined, the radix character defaults to a period (<code>.</code>).</p> <p>In other than the POSIX locale, other implementation-dependent subject sequence forms may be accepted.</p> <p>If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of <i>str</i> is stored in the object pointed to by <i>endptr</i>, provided that <i>endptr</i> is not a null pointer.</p> <p>atof() The <code>atof(str)</code> function call is equivalent to <code>strtod(str , (char **)NULL)</code>.</p> |

RETURN VALUES

Upon successful completion, **strtod()** returns the converted value. If no conversion could be performed, 0 is returned and `errno` may be set to `EINVAL`.

If the correct value is outside the range of representable values, \pm HUGE is returned (according to the sign of the value), and `errno` is set to `ERANGE`. When the `-xc` or `-xa` compilation options are used, `HUGE_VAL` is returned instead of `HUGE`.

If the correct value would cause an underflow, 0 is returned and `errno` is set to `ERANGE`.

If `str` is NaN, then **atof()** returns NaN.

ERRORS

The **strtod()** function will fail if:

ERANGE The value to be returned would cause overflow or underflow. The **strtod()** function may fail if:

EINVAL No conversion could be performed.

USAGE

Because 0 is returned on error and is also a valid return on success, an application wishing to check for error situations should set `errno` to 0, then call **strtod()**, then check `errno` and if it is non-zero, assume an error has occurred.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |
| CSI | Enabled |

SEE ALSO

isspace(3C), **localeconv(3C)**, **scanf(3S)**, **setlocale(3C)**, **strtol(3C)**, **attributes(5)**

NOTES

The **strtod()** and **atof()** functions can be used safely in multithreaded applications, as long as **setlocale(3C)** is not called to change the locale.

| | |
|-------------------------------|---|
| NAME | strtol, strtoll, atol, atoll, atoi, ltostr, ulltostr – string conversion routines |
| SYNOPSIS | <pre>#include <stdlib.h> long strtol(const char * <i>str</i>, char ** <i>endptr</i>, int <i>base</i>); long long strtoll(const char * <i>str</i>, char ** <i>endptr</i>, int <i>base</i>); long atol(const char * <i>str</i>); long long atoll(const char * <i>str</i>); int atoi(const char * <i>str</i>); char * ltostr(long long <i>value</i>, char * <i>endptr</i>); char * ulltostr(unsigned long long <i>value</i>, char * <i>endptr</i>);</pre> |
| DESCRIPTION | |
| strtol() and strtoll() | <p>The strtol() function converts the initial portion of the string pointed to by <i>str</i> to a type <code>long int</code> representation.</p> <p>The strtoll() function converts the initial portion of the string pointed to by <i>str</i> to a type <code>long long</code> representation.</p> <p>Both functions first decompose the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by isspace(3C)); a subject sequence interpreted as an integer represented in some radix determined by the value of <i>base</i>; and a final string of one or more unrecognized characters, including the terminating null byte of the input string. They then attempt to convert the subject sequence to an integer and return the result.</p> <p>If the value of <i>base</i> is 0, the expected form of the subject sequence is that of a decimal constant, octal constant or hexadecimal constant, any of which may be preceded by a + or – sign. A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal constant consists of the prefix 0 optionally followed by a sequence of the digits 0 to 7 only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal digits and letters a (or A) to f (or F) with values 10 to 15 respectively.</p> <p>If the value of <i>base</i> is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by <i>base</i>, optionally preceded by a + or – sign. The letters from a (or A) to z (or Z) inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of <i>base</i> are permitted. If the value of <i>base</i> is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.</p> |

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white-space characters, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of *base* is 0, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it is used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

In other than the POSIX locale, additional implementation-dependent subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of *str* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

**atol() , atoll() and
atoi()**

Except for behavior on error, **atol()** is equivalent to:
`strtol(str, (char **)NULL, 10)`.

Except for behavior on error, **atoll()** is equivalent to:
`strtoll(str, (char **)NULL, 10)`.

Except for behavior on error, **atoi()** is equivalent to:
`(int) strtol(str, (char **)NULL, 10)`.

lltostr() and ulltostr()

The **lltostr()** function returns a pointer to the string represented by the long long *value*. The *endptr* argument is assumed to point to the byte following a storage area into which the decimal representation of *value* is to be placed as a string. The **lltostr()** function converts *value* to decimal and produces the string, and returns a pointer to the beginning of the string. No leading zeros are produced, and no terminating null is produced. The low-order digit of the result always occupies memory position *endptr* - 1. The behavior of **lltostr()** is undefined if *value* is negative. A single zero digit is produced if *value* is 0.

The **ulltostr()** function is similar to **lltostr()** except that *value* is an unsigned long long.

RETURN VALUES

Upon successful completion, **strtol()**, **strtoll()**, **atol()**, **atoll()**, and **atoi()** return the converted value, if any. If no conversion could be performed, **strtol()** and **strtoll()** return 0 and `errno` may be set to `EINVAL`.

If the correct value is outside the range of representable values, **strtol()** returns `LONG_MAX` or `LONG_MIN` and **strtoll()** returns `LLONG_MAX` or `LLONG_MIN` (according to the sign of the value), and `errno` is set to `ERANGE`.

Upon successful completion, **ltostr()** and **ulltostr()** return a pointer to the converted string.

ERRORS

The **strtol()** and **strtoll()** functions will fail if:

ERANGE The value to be returned is not representable. The **strtol()** and **strtoll()** functions may fail if:

EINVAL The value of *base* is not supported.

USAGE

Because `0`, `LONG_MIN`, `LONG_MAX`, `LLONG_MIN`, and `LLONG_MAX` are returned on error and are also valid returns on success, an application wishing to check for error situations should set `errno` to `0`, call the function, then check `errno` and if it is non-zero, assume an error has occurred.

The **strtol()** function no longer accepts values greater than `LONG_MAX` or `LLONG_MAX` as valid input. Use **strtoul(3C)** instead.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

isalpha(3C), **isspace(3C)**, **scanf(3S)**, **strtod(3C)**, **strtoul(3C)**, **attributes(5)**

| | |
|--------------------|--|
| NAME | strtol, strtoull – convert string to unsigned long |
| SYNOPSIS | <pre>#include <stdlib.h> unsigned long strtol(const char * str, char ** endptr, int base); unsigned long long strtoull(const char * str, char ** endptr, int base);</pre> |
| DESCRIPTION | <p>The strtol() function converts the initial portion of the string pointed to by <i>str</i> to a type <code>unsigned long int</code> representation. First it decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by isspace(3C)); a subject sequence interpreted as an integer represented in some radix determined by the value of <i>base</i>; and a final string of one or more unrecognised characters, including the terminating null byte of the input string. Then it attempts to convert the subject sequence to an unsigned integer, and returns the result.</p> <p>If the value of <i>base</i> is 0, the expected form of the subject sequence is that of a decimal constant, octal constant or hexadecimal constant, any of which may be preceded by a + or – sign. A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal constant consists of the prefix 0 optionally followed by a sequence of the digits 0 to 7 only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal digits and letters a (or A) to f (or F) with values 10 to 15 respectively.</p> <p>If the value of <i>base</i> is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by <i>base</i>, optionally preceded by a + or – sign. The letters from a (or A) to z (or Z) inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of <i>base</i> are permitted. If the value of <i>base</i> is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.</p> <p>The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white-space characters, or if the first non-white-space character is other than a sign or a permissible letter or digit.</p> <p>If the subject sequence has the expected form and the value of <i>base</i> is 0, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of <i>base</i> is between 2 and 36, it is used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in the object pointed to by <i>endptr</i>, provided that <i>endptr</i> is not a null pointer.</p> |

In other than the POSIX locale, additional implementation-dependent subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of *str* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

The **strtoull()** function is identical to **strtoul()** except that it returns the value represented by *str* as an unsigned long long.

RETURN VALUES

Upon successful completion **strtoul()** returns the converted value, if any. If no conversion could be performed, 0 is returned and *errno* may be set to *EINVAL*. If the correct value is outside the range of representable values, *ULONG_MAX* is returned and *errno* is set to *ERANGE*.

ERRORS

The **strtoul()** function will fail if:

EINVAL The value of *base* is not supported.

ERANGE The value to be returned is not representable.

The **strtoul()** function may fail if:

EINVAL No conversion could be performed.

USAGE

Because 0 and *ULONG_MAX* are returned on error and are also valid returns on success, an application wishing to check for error situations should set *errno* to 0, then call **strtoul()**, then check *errno* and if it is non-zero, assume an error has occurred.

Unlike **strtod(3C)** and **strtol(3C)**, **strtoul()** must always return a non-negative number; so, using the return value of **strtoul()** for out-of-range numbers with **strtoul()** could cause more severe problems than just loss of precision if those numbers can ever be negative.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

isalpha(3C), **isspace(3C)**, **scanf(3S)**, **strtod(3C)**, **strtol(3C)**, **attributes(5)**

| | |
|----------------------|---|
| NAME | strtows, wstostr – code conversion for Process Code and File Code |
| SYNOPSIS | <pre>#include <wchar.h> wchar_t * strtows(wchar_t * dst, const char * src); char * wstostr(char * dst, const wchar_t * src);</pre> |
| DESCRIPTION | <p>The strtows() and wstostr() functions convert strings back and forth between File Code representation and Process Code.</p> <p>The strtows() function takes a character string <i>src</i> , converts it to a Process Code string, terminated by a Process Code null, and places the result into <i>dst</i> .</p> <p>The wstostr() function takes the Process Code string pointed to by <i>src</i> , converts it to a character string, and places the result into <i>dst</i> .</p> |
| RETURN VALUES | <p>The strtows() function returns the Process Code string if it completes successfully. Otherwise, a null pointer will be returned and <code>errno</code> will be set to <code>EILSEQ</code> .</p> <p>The wstostr() function returns the File Code string if it completes successfully. Otherwise, a null pointer will be returned and <code>errno</code> will be set to <code>EILSEQ</code> .</p> |
| SEE ALSO | wstring(3C) |

| | |
|----------------------|--|
| NAME | strxfrm – string transformation |
| SYNOPSIS | <pre>#include <string.h> size_t strxfrm(char *s1, const char *s2, size_t n);</pre> |
| DESCRIPTION | <p>The strxfrm() function transforms the string pointed to by <i>s2</i> and places the resulting string into the array pointed to by <i>s1</i>. The transformation is such that if strcmp(3C) is applied to two transformed strings, it returns a value greater than, equal to or less than 0, corresponding to the result of strcoll(3C) applied to the same two original strings. No more than <i>n</i> bytes are placed into the resulting array pointed to by <i>s1</i>, including the terminating null byte. If <i>n</i> is 0, <i>s1</i> is permitted to be a null pointer. If copying takes place between objects that overlap, the behavior is undefined.</p> |
| RETURN VALUES | <p>Upon successful completion, strxfrm() returns the length of the transformed string (not including the terminating null byte). If the value returned is <i>n</i> or more, the contents of the array pointed to by <i>s1</i> are indeterminate.</p> <p>On failure, strxfrm() returns <code>(size_t)-1</code>.</p> |
| USAGE | <p>The transformation function is such that two transformed strings can be ordered by strcmp(3C) as appropriate to collating sequence information in the program's locale (category <code>LC_COLLATE</code>).</p> <p>The fact that when <i>n</i> is 0, <i>s1</i> is permitted to be a null pointer, is useful to determine the size of the <i>s1</i> array prior to making the transformation.</p> <p>Because no return value is reserved to indicate an error, an application wishing to check for error situations should set <code>errno</code> to 0, then call strcoll(3C), then check <code>errno</code> and if it is non-zero, assume an error has occurred.</p> <p>This issue is aligned with the ANSI C standard; this does not affect compatibility with XPG3 applications. Reliable error detection by this function was never guaranteed.</p> |
| EXAMPLES | <p>EXAMPLE 1 A sample of using the strxfrm() function.</p> <p>The value of the following expression is the size of the array needed to hold the transformation of the string pointed to by <i>s</i>.</p> <pre>1 + strxfrm(NULL, s, 0);</pre> |
| FILES | <pre>/usr/lib/locale/<i>locale/locale</i>.so.*</pre> <p style="text-align: right;"><code>LC_COLLATE</code> database for <i>locale</i></p> |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |
| CSI | Enabled |

SEE ALSO

localedef(1), **setlocale(3C)**, **strcmp(3C)**, **strcoll(3C)**, **wscoll(3C)**, **attributes(5)**, **environ(5)**, **standards(5)**

NOTES

The **strxfrm()** function can be used safely in a multithreaded application, as long as **setlocale(3C)** is not being called to change the locale.

NAME swab – swap bytes

SYNOPSIS #include <unistd.h>
 void **swab**(const void *src, void *dest, ssize_t nbytes);

DESCRIPTION The **swab()** function copies *nbytes* bytes, which are pointed to by *src*, to the object pointed to by *dest*, exchanging adjacent bytes. The *nbytes* argument should be even. If *nbytes* is odd **swab()** copies and exchanges *nbytes*–1 bytes and the disposition of the last byte is unspecified. If copying takes place between objects that overlap, the behavior is undefined. If *nbytes* is negative, **swab()** does nothing.

ERRORS No errors are defined.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **attributes(5)**

| NAME | sync_instruction_memory - make modified instructions executable | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | void sync_instruction_memory(caddr_t addr, int len); | | | | |
| DESCRIPTION | <p>The sync_instruction_memory() function performs whatever steps are required to make instructions modified by a program executable.</p> <p>Some processor architectures, including some SPARC processors, have separate and independent instruction and data caches which are not kept consistent by hardware. For example, if the instruction cache contains an instruction from some address and the program then stores a new instruction at that address, the new instruction may not be immediately visible to the instruction fetch mechanism. Software must explicitly invalidate the instruction cache entries for new or changed mappings of pages that might contain executable instructions. The sync_instruction_memory() function performs this function, and/or any other functions needed to make modified instructions between <i>addr</i> and <i>addr+len</i> visible. A program should call sync_instruction_memory() after modifying instructions and before executing them.</p> <p>On processors with unified caches (one cache for both instructions and data) and pipelines which are flushed by a branch instruction, such as the Intel x86 architecture, the function may do nothing and just return.</p> <p>The changes are immediately visible to the thread calling sync_instruction_memory() when the call returns, even if the thread should migrate to another processor during or after the call. The changes become visible to other threads in the same manner that stores do; that is, they eventually become visible, but the latency is implementation-dependent.</p> <p>The result of executing sync_instruction_memory() are unpredictable if <i>addr</i> through <i>addr+len-1</i> are not valid for the address space of the program making the call.</p> | | | | |
| RETURN VALUES | No values are returned. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | attributes(5) | | | | |

| | |
|----------------------|---|
| NAME | syncok, wcursyncup, wsyncdown, wsyncup – synchronize window with its parents or children |
| SYNOPSIS | <pre>#include <curses.h> int syncok(WINDOW * win, bool bf); void wcursyncup(WINDOW * win); void wsyncdown(WINDOW * win); void wsyncup(WINDOW * win);</pre> |
| PARAMETERS | <p>win Is a pointer to a window.</p> <p>bf Is a Boolean expression.</p> |
| DESCRIPTION | <p>The syncok() function uses the value of <i>bf</i> to determine whether or not the window <i>win</i> 's ancestors are implicitly touched whenever there is a change to <i>win</i> . If <i>bf</i> is TRUE , this touching occurs. If <i>bf</i> is FALSE , it does not occur. The initial value for <i>bf</i> is FALSE .</p> <p>The wcursyncup() function moves the cursor in <i>win</i> 's ancestors to match its position in <i>win</i> .</p> <p>The wsyncdown() function touches <i>win</i> if any of its ancestors have been touched.</p> <p>The wsyncup() function touches all ancestors of <i>win</i> .</p> |
| RETURN VALUES | <p>On success, the syncok() function returns OK . Otherwise, it returns ERR .</p> <p>The other functions do not return a value.</p> |
| ERRORS | None. |
| SEE ALSO | derwin(3XC) , doupdate(3XC) , is_linetouched(3XC) |

| | |
|----------------------|---|
| NAME | syscall – indirect system call |
| SYNOPSIS | <pre>/usr/ucb/cc [<i>flag ...</i>] <i>file ...</i> #include <sys/syscall.h> int syscall(<i>number, arg, ...</i>);</pre> |
| DESCRIPTION | syscall() performs the function whose assembly language interface has the specified <i>number</i> , and arguments <i>arg ...</i> . Symbolic constants for functions can be found in the header <sys/syscall.h>. |
| RETURN VALUES | On error syscall() returns -1 and sets the external variable <i>errno</i> (see intro(2)). |
| FILES | <sys/syscall.h> |
| SEE ALSO | intro(2) , pipe(2) |
| NOTES | Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported. |
| WARNINGS | There is no way to use syscall() to call functions such as pipe(2) which return values that do not fit into one hardware register. Since many system calls are implemented as library wrappers around traps to the kernel, these calls may not behave as documented when called from syscall() , which bypasses these wrappers. For these reasons, using syscall() is not recommended. |

NAME sysconf – get configurable system variables

SYNOPSIS #include <unistd.h>

long **sysconf**(int *name*);

DESCRIPTION

The **sysconf()** function provides a method for an application to determine the current value of a configurable system limit or option (variable).

The *name* argument represents the system variable to be queried. The following table lists the minimal set of system variables from <limits.h> and <unistd.h> that can be returned by **sysconf()** and the symbolic constants defined in <unistd.h> that are the corresponding values used for *name* on the SPARC and x86 platforms.

| Name | Return Value | Meaning |
|-----------------|-------------------|---|
| _SC_2_C_BIND | _POSIX2_C_BIND | Supports the C language binding option |
| _SC_2_C_DEV | _POSIX2_C_DEV | Supports the C language development utilities option |
| _SC_2_C_VERSION | _POSIX2_C_VERSION | Integer value indicating version of ISO POSIX-2 standard (Commands) |
| _SC_2_CHAR_TERM | _POSIX2_CHAR_TERM | Supports at least one terminal |
| _SC_2_FORT_DEV | _POSIX2_FORT_DEV | Supports FORTRAN Development Utilities Option |
| _SC_2_FORT_RUN | _POSIX2_FORT_RUN | Supports FORTRAN Run-time Utilities Option |
| _SC_2_LOCALEDEF | _POSIX2_LOCALEDEF | Supports the creation of locales by the localedef utility |
| _SC_2_SW_DEV | _POSIX2_SW_DEV | Supports the Software Development Utility Option |
| _SC_2_UPE | _POSIX2_UPE | Supports the User Portability Utilities Option |

| Name | Return Value | Meaning |
|------------------------|------------------------|---|
| _SC_2_VERSION | _POSIX2_VERSION | Integer value indicating version of ISO POSIX-2 standard (C language binding) |
| _SC_AIO_LISTIO_MAX | AIO_LISTIO_MAX | Max number of I/O operations in a single list I/O call supported |
| _SC_AIO_MAX | AIO_MAX | Max number of outstanding asynchronous I/O operations supported |
| _SC_AIO_PRIO_DELTA_MAX | AIO_PRIO_DELTA_MAX | Max amount by which a process can decrease its asynchronous I/O priority level from its own scheduling priority |
| _SC_ARG_MAX | ARG_MAX | Max size of <code>argv[]</code> plus <code>envp[]</code> |
| _SC_ASYNCHRONOUS_IO | _POSIX_ASYNCHRONOUS_IO | Supports Asynchronous I/O |
| _SC_ATEXIT_MAX | ATEXIT_MAX | Max number of functions that may be registered with <code>atexit()</code> |
| _SC_AVPHYS_PAGES | | Number physical memory pages not currently in use by system |
| _SC_BC_BASE_MAX | BC_BASE_MAX | Maximum <i>obase</i> values allowed by the <code>bc</code> utility |
| _SC_BC_DIM_MAX | BC_DIM_MAX | Max number of elements permitted in array by <code>bc</code> |
| _SC_BC_SCALE_MAX | BC_SCALE_MAX | Max <i>scale</i> value allowed by <code>bc</code> |
| _SC_BC_STRING_MAX | BC_STRING_MAX | Max length of string constant allowed by <code>bc</code> |
| _SC_CHILD_MAX | CHILD_MAX | Max processes allowed to a UID |
| _SC_CLK_TCK | CLK_TCK | Ticks per second (<code>clock_t</code>) |

| Name | Return Value | Meaning |
|-----------------------|--------------------------|--|
| _SC_COLL_WEIGHTS_MAX | COLL_WEIGHTS_MAX | Max number of weights that can be assigned to entry of the LC_COLLATE order keyword in locale definition file |
| _SC_DELAYTIMER_MAX | DELAYTIMER_MAX | Max number of timer expiration overruns |
| _SC_EXPR_NEST_MAX | EXPR_NEST_MAX | Max number of expressions that can be listed within parentheses by the <code>expr</code> utility |
| _SC_FSYNC | _POSIX_FSYNC | Supports File Synchronization |
| _SC_GETGR_R_SIZE_MAX | NSS_BUFLLEN_GROUP | Max size of group entry buffer |
| _SC_GETPW_R_SIZE_MAX | NSS_BUFLLEN_PASSWD | Max size of password entry buffer |
| _SC_IOV_MAX | IOV_MAX | Max number of <code>iovec</code> structures that one process has available for use with <code>readv()</code> and <code>writev()</code> |
| _SC_JOB_CONTROL | _POSIX_JOB_CONTROL | Job control supported |
| _SC_LINE_MAX | LINE_MAX | Max length of input line |
| _SC_LOGIN_NAME_MAX | LOGNAME_MAX + 1 | Max length of login name. |
| _SC_LOGNAME_MAX | LOGNAME_MAX | |
| _SC_MAPPED_FILES | _POSIX_MAPPED_FILES | Supports Memory Mapped Files |
| _SC_MEMLOCK | _POSIX_MEMLOCK | Supports Process Memory Locking. |
| _SC_MEMLOCK_RANGE | _POSIX_MEMLOCK_RANGE | Supports Range Memory Locking. |
| _SC_MEMORY_PROTECTION | _POSIX_MEMORY_PROTECTION | Supports Memory Protection. |
| _SC_MESSAGE_PASSING | _POSIX_MESSAGE_PASSING | Supports Message Passing. |

| Name | Return Value | Meaning |
|-------------------------|----------------------------|---|
| _SC_MQ_OPEN_MAX | MQ_OPEN_MAX | Max number of open message queues a process may hold. |
| _SC_MQ_PRIO_MAX | MQ_PRIO_MAX | Max number of message priorities supported |
| _SC_NGROUPS_MAX | NGROUPS_MAX | Max simultaneous groups to which one may belong |
| _SC_NPROCESSORS_CONF | | Number of processors configured |
| _SC_NPROCESSORS_ONLN | | Number of processors online |
| _SC_OPEN_MAX | OPEN_MAX | Max open files per process |
| _SC_PAGESIZE | PAGESIZE | System memory page size |
| _SC_PAGE_SIZE | PAGESIZE | Same as _SC_PAGESIZE |
| _SC_PASS_MAX | PASS_MAX | Max number of significant bytes in a password |
| _SC_PHYS_PAGES | | Total number of pages of physical memory in system |
| _SC_PRIORITIZED_IO | _POSIX_PRIORITIZED_IO | Supports Prioritized I/O |
| _SC_PRIORITY_SCHEDULING | _POSIX_PRIORITY_SCHEDULING | Supports Process Scheduling |
| _SC_RE_DUP_MAX | RE_DUP_MAX | Max number of repeated |
| _SC_REALTIME_SIGNALS | _POSIX_REALTIME_SIGNALS | Supports Realtime Signals. |
| _SC_RTSIG_MAX | RTSIG_MAX | Max number of realtime signals reserved for application use |
| _SC_SAVED_IDS | _POSIX_SAVED_IDS | Saved IDs (seteuid()) supported? |
| _SC_SEM_NSEMS_MAX | SEM_NSEMS_MAX | Max number of semaphores that a process may have. |
| _SC_SEM_VALUE_MAX | SEM_VALUE_MAX | Max value a semaphore may have |

| Name | Return Value | Meaning |
|----------------------------------|-----------------------------------|--|
| _SC_SEMAPHORES | _POSIX_SEMAPHORES | Supports Semaphores |
| _SC_SHARED_MEMORY_OBJECTS | _POSIX_SHARED_MEMORY_OBJECTS | Supports Shared MemoryObjects |
| _SC_SIGQUEUE_MAX | SIGQUEUE_MAX | Max number of queued signals a process may send and have pending at receiver(s) at a time. |
| _SC_STREAM_MAX | STREAM_MAX | Number of streams one processed can have open at a time |
| _SC_SYNCHRONIZED_IO | _POSIX_SYNCHRONIZED_IO | Supports Synchronized I/O. |
| _SC_THREAD_ATTR_STACKADDR | _POSIX_THREAD_ATTR_STACKADDR | Supports Thread Stack Address Attribute option |
| _SC_THREAD_ATTR_STACKSIZE | _POSIX_THREAD_ATTR_STACKSIZE | Supports Thread Stack Size Attribute option |
| _SC_THREAD_DESTRUCTOR_ITERATIONS | PTHREAD_DESTRUCTOR_ITERATIONS | Number attempts made to destroy thread-specific data on thread exit |
| _SC_THREAD_KEYS_MAX | PTHREAD_KEYS_MAX | Max number of data keys per process |
| _SC_THREAD_PRIO_INHERIT | _POSIX_THREAD_PRIO_INHERIT | Supports Priority Inheritance option |
| _SC_THREAD_PRIO_PROTECT | _POSIX_THREAD_PRIO_PROTECT | Supports Priority Protection option |
| _SC_THREAD_PRIORITY_SCHEDULING | _POSIX_THREAD_PRIORITY_SCHEDULING | Supports Thread Execution Scheduling option |
| _SC_THREAD_PROCESS_SHARED | _POSIX_THREAD_PROCESS_SHARED | Supports Process-Shared Synchronization option |
| _SC_THREAD_SAFE_FUNCTIONS | _POSIX_THREAD_SAFE_FUNCTIONS | Supports Thread-Safe Functions option |
| _SC_THREAD_STACK_MIN | PTHREAD_STACK_MIN | Min byte size of thread stack storage |

| Name | Return Value | Meaning |
|------------------------|---------------------|--|
| _SC_THREAD_THREADS_MAX | PTHREAD_THREADS_MAX | Max number of threads per process |
| _SC_THREADS | _POSIX_THREADS | Supports Threads option |
| _SC_TIMER_MAX | TIMER_MAX | Max number of timers per process |
| _SC_TIMERS | _POSIX_TIMERS | Supports Timers |
| _SC_TTY_NAME_MAX | TTYNAME_MAX | Max length of tty device name |
| _SC_TZNAME_MAX | TZNAME_MAX | Max number of bytes supported for name of a time zone |
| _SC_VERSION | _POSIX_VERSION | POSIX.1 version supported occurrences of a regular expression permitted when using the interval notation $\{m,n\}$ |
| _SC_XBS5_ILP32_OFF32 | _XBS_ILP32_OFF32 | Supports X/Open ILP32 w/32-bit offset build environment |
| _SC_XBS5_ILP32_OFFBIG | _XBS5_ILP32_OFFBIG | Supports X/Open LP32 w/64-bit offset build environment |
| _SC_XBS5_LP64_OFF64 | _XBS5_LP64_OFF64 | Supports X/Open LP64, 64-bit offset build environment |
| _SC_XBS5_LPBIG_OFFBIG | _XBS5_LP64_OFF64 | Same as _SC_XBS5_LP64_OFF64 |
| _SC_XOPEN_CRYPT | _XOPEN_CRYPT | Supports X/Open Encryption Feature Group |
| _SC_XOPEN_ENH_I18N | _XOPEN_ENH_I18N | Supports X/Open Enhanced Internationalization Feature Group |
| _SC_XOPEN_LEGACY | _XOPEN_LEGACY | Supports X/Open Legacy Feature Group |
| _SC_XOPEN_REALTIME | _XOPEN_REALTIME | Supports X/Open POSIX Realtime Feature Group |

| Name | Return Value | Meaning |
|---|--------------------------------------|---|
| <code>_SC_XOPEN_REALTIME_THREADS</code> | <code>_XOPEN_REALTIME_THREADS</code> | Supports X/Open POSIX Realtime Threads Feature Group |
| <code>_SC_XOPEN_SHM</code> | <code>_XOPEN_SHM</code> | Supports X/Open Shared Memory Feature Group |
| <code>_SC_XOPEN_UNIX</code> | <code>_XOPEN_UNIX</code> | Supports X/Open CAE Specification, August 1994, System Interfaces and Headers, Issue 4, Version 2 |
| <code>_SC_XOPEN_VERSION</code> | <code>_XOPEN_VERSION</code> | Integer value indicating version of X/Open Portability Guide to which implementation conforms |
| <code>_SC_XOPEN_XCU_VERSION</code> | <code>_XOPEN_XCU_VERSION</code> | Integer value indicating version of XCU specification to which implementation conforms |

RETURN VALUES

Upon successful completion, **sysconf()** returns the current variable value on the system. The value returned will not be more restrictive than the corresponding value described to the application when it was compiled with the implementation's `<limits.h>`, `<unistd.h>` or `<time.h>`. The value will not change during the lifetime of the calling process.

If *name* is an invalid value, **sysconf()** returns `-1` and sets `errno` to indicate the error. If the variable corresponding to *name* is associated with functionality that is not supported by the system, **sysconf()** returns `-1` without changing the value of `errno`.

Calling **sysconf()** with `_SC_THREAD_KEYS_MAX`, `_SC_THREAD_THREADS_MAX`, or `_SC_THREAD_DESTRUCTOR_ITERATIONS` returns `-1` without setting `errno`, because no maximum limit can be determined. The system supports at least the minimum values defined by `_POSIX_THREAD_KEYS_MAX`, `_POSIX_THREAD_THREADS_MAX`, and `_POSIX_THREAD_DESTRUCTOR_ITERATIONS` and can support higher values depending upon system resources.

The following SPARC and x86 platform variables return `EINVAL`:

| | |
|-------------------|-------------------|
| _SC_COHER_BLKSZ | _SC_DCACHE_ASSOC |
| _SC_DCACHE_BLKSZ | _SC_DCACHE_LINESZ |
| _SC_DCACHE_SZ | _SC_DCACHE_TBLKSZ |
| _SC_ICACHE_ASSOC | _SC_ICACHE_BLKSZ |
| _SC_ICACHE_LINESZ | _SC_ICACHE_SZ |
| _SC_SPLIT_CACHE | |

ERRORS

The **sysconf()** function will fail if:

EINVAL The value of the *name* argument is invalid.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|----------------------------|
| Architecture | SPARC and x86 |
| MT-Level | MT-Safe, Async-Signal-Safe |

SEE ALSO

fpathconf(2), **seteuid(2)**, **setrlimit(2)**, **attributes(5)**, **standards(5)**

NOTES

A call to **setrlimit()** may cause the value of `OPEN_MAX` to change.

Multiplying `sysconf(_SC_PHYS_PAGES)` or `sysconf(_SC_AVPHYS_PAGES)` by `sysconf(_SC_PAGESIZE)` to determine memory amount in bytes can exceed the maximum values representable in a long or unsigned long.

The value of `CLK_TCK` may be variable and it should not be assumed that `CLK_TCK` is a compile-time constant.

The `_SC_PHYS_PAGES` and `_SC_AVPHYS_PAGES` variables are specific to Solaris 2.3 or compatible releases.

The `_SC_THREAD_PRIO_INHERIT` and `_SC_THREAD_PRIO_PROTECT` variables are currently not supported. A call to **sysconf()** with these variables as arguments returns `-1`.

| | | | | | | | | | | | | | | | |
|--------------------|---|-----------|---|-----------|--|----------|--|---------|---------|-------------|-------------------|------------|--|----------|-------------------------|
| NAME | syslog, openlog, closelog, setlogmask – control system log | | | | | | | | | | | | | | |
| SYNOPSIS | <pre>#include <syslog.h> void openlog(const char * <i>ident</i>, int <i>logopt</i>, int <i>facility</i>); void syslog(int <i>priority</i>, const char * <i>message</i>, .../* <i>arguments</i> */); void closelog(void); int setlogmask(int <i>maskpri</i>);</pre> | | | | | | | | | | | | | | |
| DESCRIPTION | <p>The syslog() function sends a message to syslogd(1M), which, depending on the configuration of <code>/etc/syslog.conf</code>, logs it in an appropriate system log, writes it to the system console, forwards it to a list of users, or forwards it to syslogd on another host over the network. The logged message includes a message header and a message body. The message header consists of a facility indicator, a severity level indicator, a timestamp, a tag string, and optionally the process ID.</p> <p>The message body is generated from the <i>message</i> and following arguments in the same manner as if these were arguments to printf(3B), except that occurrences of <code>%m</code> in the format string pointed to by the <i>message</i> argument are replaced by the error message string associated with the current value of <code>errno</code>. A trailing NEWLINE character is added if needed.</p> <p>Values of the <i>priority</i> argument are formed by ORing together a <i>severity level</i> value and an optional <i>facility</i> value. If no facility value is specified, the current default facility value is used.</p> <p>Possible values of severity level include:</p> <table border="0"> <tr> <td style="padding-right: 20px;">LOG_EMERG</td> <td>A panic condition. This is normally broadcast to all users.</td> </tr> <tr> <td>LOG_ALERT</td> <td>A condition that should be corrected immediately, such as a corrupted system database.</td> </tr> <tr> <td>LOG_CRIT</td> <td>Critical conditions, such as hard device errors.</td> </tr> <tr> <td>LOG_ERR</td> <td>Errors.</td> </tr> <tr> <td>LOG_WARNING</td> <td>Warning messages.</td> </tr> <tr> <td>LOG_NOTICE</td> <td>Conditions that are not error conditions, but that may require special handling.</td> </tr> <tr> <td>LOG_INFO</td> <td>Informational messages.</td> </tr> </table> | LOG_EMERG | A panic condition. This is normally broadcast to all users. | LOG_ALERT | A condition that should be corrected immediately, such as a corrupted system database. | LOG_CRIT | Critical conditions, such as hard device errors. | LOG_ERR | Errors. | LOG_WARNING | Warning messages. | LOG_NOTICE | Conditions that are not error conditions, but that may require special handling. | LOG_INFO | Informational messages. |
| LOG_EMERG | A panic condition. This is normally broadcast to all users. | | | | | | | | | | | | | | |
| LOG_ALERT | A condition that should be corrected immediately, such as a corrupted system database. | | | | | | | | | | | | | | |
| LOG_CRIT | Critical conditions, such as hard device errors. | | | | | | | | | | | | | | |
| LOG_ERR | Errors. | | | | | | | | | | | | | | |
| LOG_WARNING | Warning messages. | | | | | | | | | | | | | | |
| LOG_NOTICE | Conditions that are not error conditions, but that may require special handling. | | | | | | | | | | | | | | |
| LOG_INFO | Informational messages. | | | | | | | | | | | | | | |

| | |
|---|--|
| LOG_DEBUG | Messages that contain information normally of use only when debugging a program. |
| The facility indicates the application or system component generating the message. Possible facility values include: | |
| LOG_KERN | Messages generated by the kernel. These cannot be generated by any user processes. |
| LOG_USER | Messages generated by random user processes. This is the default facility identifier if none is specified. |
| LOG_MAIL | The mail system. |
| LOG_DAEMON | System daemons, such as <code>in.ftpd(1M)</code> . |
| LOG_AUTH | The authorization system: <code>login(1)</code> , <code>su(1M)</code> , <code>getty(1M)</code> . |
| LOG_LPR | The line printer spooling system: <code>lpr(1B)</code> , <code>lpc(1B)</code> . |
| LOG_NEWS | Reserved for the USENET network news system. |
| LOG_UUCP | Reserved for the UUCP system; it does not currently use <code>syslog</code> . |
| LOG_CRON | The <code>cron</code> / <code>at</code> facility; <code>crontab(1)</code> , <code>at(1)</code> , <code>cron(1M)</code> . |
| LOG_LOCAL0 | Reserved for local use. |
| LOG_LOCAL1 | Reserved for local use. |
| LOG_LOCAL2 | Reserved for local use. |
| LOG_LOCAL3 | Reserved for local use. |
| LOG_LOCAL4 | Reserved for local use. |
| LOG_LOCAL5 | Reserved for local use. |
| LOG_LOCAL6 | Reserved for local use. |
| LOG_LOCAL7 | Reserved for local use. |
| The openlog() function sets process attributes that affect subsequent calls to syslog() . The <i>ident</i> argument is a string that is prepended to every message. | |

The *logopt* argument indicates logging options. Values for *logopt* are constructed by a bitwise-inclusive OR of zero or more of the following:

| | |
|------------|--|
| LOG_PID | Log the process ID with each message. This is useful for identifying specific daemon processes (for daemons that fork). |
| LOG_CONS | Write messages to the system console if they cannot be sent to <code>syslogd(1M)</code> . This option is safe to use in daemon processes that have no controlling terminal, since <code>syslog()</code> forks before opening the console. |
| LOG_NDELAY | Open the connection to <code>syslogd(1M)</code> immediately. Normally the open is delayed until the first message is logged. This is useful for programs that need to manage the order in which file descriptors are allocated. |
| LOG_ODELAY | Delay open until <code>syslog()</code> is called. |
| LOG_NOWAIT | Do not wait for child processes that have been forked to log messages onto the console. This option should be used by processes that enable notification of child termination using <code>SIGCHLD</code> , since <code>syslog()</code> may otherwise block waiting for a child whose exit status has already been collected. |

The *facility* argument encodes a default facility to be assigned to all messages that do not have an explicit facility already encoded. The initial default facility is `LOG_USER`.

The `openlog()` and `syslog()` functions may allocate a file descriptor. It is not necessary to call `openlog()` prior to calling `syslog()`.

The `closelog()` function closes any open file descriptors allocated by previous calls to `openlog()` or `syslog()`.

The `setlogmask()` function sets the log priority mask for the current process to *maskpri* and returns the previous mask. If the *maskpri* argument is 0, the current log mask is not modified. Calls by the current process to `syslog()` with a priority not set in *maskpri* are rejected. The mask for an individual priority *pri* is calculated by the macro `LOG_MASK(pri)`; the mask for all priorities up to and including *toppri* is given by the macro `LOG_UPT(toppri)`. The default log mask allows all priorities to be logged.

Symbolic constants for use as values of the *logopt*, *facility*, *priority*, and *maskpri* arguments are defined in the `< syslog.h >` header.

RETURN VALUES

The **setlogmask()** function returns the previous log priority mask. The **closelog()**, **openlog()** and **syslog()** functions return no value.

ERRORS

No errors are defined.

EXAMPLES

EXAMPLE 1 Example of LOG_ALERT message.

This call logs a message at priority LOG_ALERT :

```
syslog(LOG_ALERT, "who: internal error 23")
;
```

The FTP daemon `ftpd` would make this call to **openlog()** to indicate that all messages it logs should have an identifying string of `ftpd`, should be treated by `syslogd(1M)` as other messages from system daemons are, should include the process ID of the process logging the message:

```
openlog("ftpd", LOG_PID, LOG_DAEMON)
;
```

Then it would make the following call to **setlogmask()** to indicate that messages at priorities from LOG_EMERG through LOG_ERR should be logged, but that no messages at any other priority should be logged:

```
setlogmask(LOG_UPTO(LOG_ERR))
;
```

Then, to log a message at priority LOG_INFO, it would make the following call to `syslog` :

```
syslog(LOG_INFO, "Connection from host %d", CallingHost)
;
```

A locally-written utility could use the following call to **syslog()** to log a message at priority LOG_INFO to be treated by `syslogd(1M)` as other messages to the facility LOG_LOCAL2 are:

```
syslog(LOG_INFO|LOG_LOCAL2, "error: %m")
;
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

at(1), **crontab(1)**, **logger(1)**, **login(1)**, **lpc(1B)**, **lpr(1B)**, **cron(1M)**, **getty(1M)**, **in.ftpd(1M)**, **su(1M)**, **syslogd(1M)**, **printf(3B)**, **syslog.conf(4)**, **attributes(5)**

| | |
|----------------------|--|
| NAME | systemem, asystemem – return physical memory information |
| SYNOPSIS | long systemem (void); long asystemem (void); |
| DESCRIPTION | <p>These routines are obsolete and have been replaced by arguments to sysconf(3C) . They were mistakenly published in the <i>System V Interface Definition</i> , Third Edition, (SVID) and corrected by the Errata: "The following routines were mistakenly include in SVID Edition 3 and were not designed as customer level interfaces: <code>systemem(AS_LIB)</code> , <code>asystemem(AS_LIB)</code> , ... They are therefore removed."</p> <p>The routine systemem() determines the total amount of physical memory of the system. It returns a long integer representing the total amount of physical memory, in bytes. Because systemem() returns a long integer it cannot report the amount of memory for configurations with amounts of memory in bytes greater than the maximum positive value represented by a long integer. <code>sysconf(_SC_PHYS_PAGES)</code> should be used to avoid this limitation. (See sysconf(3C) .)</p> <p>The routine asystemem() determines the total amount of memory not currently in use on the system. It returns a long integer representing the total amount of available memory, in bytes. Because asystemem() returns a long integer it is limited similar to systemem() . <code>sysconf(_SC_AVPHYS_PAGES)</code> should be used to avoid this limitation. (See sysconf(3C) .)</p> |
| RETURN VALUES | Upon successful completion, these routines return the amount of memory in bytes; otherwise, they return -1 . |
| SEE ALSO | sysconf (3C) |
| NOTES | systemem () and asystemem () are obsolete and should be replaced with sysconf (3C) . |

| NAME | system – issue a shell command | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | #include <stdlib.h> int system (const char *string); | | | | |
| DESCRIPTION | The system() function causes <i>string</i> to be given to the shell as input, as if <i>string</i> had been typed as a command at a terminal. The invoker waits until the shell has completed, then returns the exit status of the shell in the format specified by waitpid(2) . If <i>string</i> is a null pointer, system() checks if the shell exists and is executable. If the shell is available, system() returns a non-zero value; otherwise, it returns 0. If the application is standard-conforming (see standards(5)), system() uses /usr/bin/ksh (see ksh(1)); otherwise system() uses /usr/bin/sh (see sh(1)). | | | | |
| RETURN VALUES | The system() function forks to create a child process that in turn invokes one of the exec family of functions (see exec(2)) on the shell to execute <i>string</i> . If fork() or the exec function fails, system() returns -1 and sets errno to indicate the error. | | | | |
| ERRORS | The system() function fails if: EAGAIN The system-imposed limit on the total number of processes under execution by a single user would be exceeded. EINTR The system() function was interrupted by a signal. ENOMEM The new process requires more memory than is available. | | | | |
| USAGE | The system() function will fail to execute setuid() or setgid() if either the UID or GID of the application's owner/group is less than 100. See useradd(1M) and setuid(2) . | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | ksh(1) , sh(1) , useradd(1M) , exec(2) , fork(2) , setuid(2) , waitpid(2) , attributes(5) , standards(5) | | | | |

| | |
|--------------------|--|
| NAME | t_accept – accept a connection request |
| SYNOPSIS | <pre>#include <xti.h></pre> <pre>int t_accept(int fd, int resfd, const struct t_call *call);</pre> |
| DESCRIPTION | <p>This routine is part of the XTI interfaces that evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, <code>tiuser.h</code>, must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>This function is issued by a transport user to accept a connection request. The parameter <i>fd</i> identifies the local transport endpoint where the connection indication arrived; <i>resfd</i> specifies the local transport endpoint where the connection is to be established, and <i>call</i> contains information required by the transport provider to complete the connection. The parameter <i>call</i> points to a <code>t_call</code> structure which contains the following members:</p> <pre>struct netbuf addr; struct netbuf opt; struct netbuf udata; int sequence;</pre> <p>In <i>call</i>, <i>addr</i> is the protocol address of the calling transport user, <i>opt</i> indicates any options associated with the connection, <i>udata</i> points to any user data to be returned to the caller, and <i>sequence</i> is the value returned by <code>t_listen(3N)</code> that uniquely associates the response with a previously received connection indication. The address of the caller, <i>addr</i> may be null (length zero). Where <i>addr</i> is not null then it may optionally be checked by XTI.</p> <p>A transport user may accept a connection on either the same, or on a different, local transport endpoint than the one on which the connection indication arrived. Before the connection can be accepted on the same endpoint (<i>resfd==fd</i>), the user must have responded to any previous connection indications received on that transport endpoint by means of <code>t_accept()</code> or <code>t_snddis(3N)</code>. Otherwise, <code>t_accept()</code> will fail and set <code>t_errno</code> to <code>TINDOUT</code>.</p> <p>If a different transport endpoint is specified (<i>resfd!=fd</i>), then the user may or may not choose to bind the endpoint before the <code>t_accept()</code> is issued. If the endpoint is not bound prior to the <code>t_accept()</code>, the endpoint must be in the <code>T_UNBND</code> state before the <code>t_accept()</code> is issued, and the transport provider will automatically bind it to an address that is appropriate for the protocol</p> |

concerned. If the transport user chooses to bind the endpoint it must be bound to a protocol address with a *qlen* of zero and must be in the `T_IDLE` state before the `t_accept()` is issued.

Responding endpoints should be supplied to `t_accept()` in the state `T_UNBND`.

The call to `t_accept()` may fail with `t_errno` set to `TLOOK` if there are indications (for example connect or disconnect) waiting to be received on endpoint *fd*. Applications should be prepared for such a failure.

The *udata* argument enables the called transport user to send user data to the caller and the amount of user data must not exceed the limits supported by the transport provider as returned in the *connect* field of the *info* argument of `t_open(3N)` or `t_getinfo(3N)`. If the *len* field of *udata* is zero, no data will be sent to the caller. All the *maxlen* fields are meaningless.

When the user does not indicate any option ($call \rightarrow opt.len = 0$) the connection shall be accepted with the option values currently set for the responding endpoint *resfd*.

RETURN VALUES

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and `t_errno` is set to indicate an error.

VALID STATES

fd: `T_INCON`

resfd ($fd \neq resfd$): `T_IDLE`, `T_UNBND`

ERRORS

On failure, `t_errno` is set to one of the following:

| | |
|-----------------------|--|
| <code>TACCES</code> | The user does not have permission to accept a connection on the responding transport endpoint or to use the specified options. |
| <code>TBADADDR</code> | The specified protocol address was in an incorrect format or contained illegal information. |
| <code>TBADDATA</code> | The amount of user data specified was not within the bounds allowed by the transport provider. |
| <code>TBADF</code> | The file descriptor <i>fd</i> or <i>resfd</i> does not refer to a transport endpoint. |
| <code>TBADOPT</code> | The specified options were in an incorrect format or contained illegal information. |
| <code>TBADSEQ</code> | Either an invalid sequence number was specified, or a valid sequence number was specified but the connection request was aborted by the peer. In |

| | |
|---------------|---|
| | the latter case, its <code>T_DISCONNECT</code> event will be received on the listening endpoint. |
| TINDOUT | The function was called with <code>fd==resfd</code> but there are outstanding connection indications on the endpoint. Those other connection indications must be handled either by rejecting them by means of <code>t_snddis(3N)</code> or accepting them on a different endpoint by means of <code>t_accept</code> . |
| TLOOK | An asynchronous event has occurred on the transport endpoint referenced by <code>fd</code> and requires immediate attention. |
| TNOTSUPPORT | This function is not supported by the underlying transport provider. |
| TOUTSTATE | The communications endpoint referenced by <code>fd</code> or <code>resfd</code> is not in one of the states in which a call to this function is valid. |
| TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). |
| TPROVMISMATCH | The file descriptors <code>fd</code> and <code>resfd</code> do not refer to the same transport provider. |
| TRESADDR | This transport provider requires both <code>fd</code> and <code>resfd</code> to be bound to the same address. This error results if they are not. |
| TRESQLEN | The endpoint referenced by <code>resfd</code> (where <code>resfd != fd</code>) was bound to a protocol address with a <code>qlen</code> that is greater than zero. |
| TSYSERR | A system error has occurred during execution of this function. |

TLI COMPATIBILITY

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```
#include <tiuser.h>
```

Error Description Values

The `t_errno` values that can be set by the XTI interface and cannot be set by the TLI interface are:

TPROTO

TINDOUT

TPROVMISMATCH

TRESADDR

TRESQLEN

Option Buffer

The format of the options in an `opt` buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not specify the buffer format.

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`t_connect(3N)`, `t_getinfo(3N)`, `t_getstate(3N)`, `t_listen(3N)`, `t_open(3N)`, `t_optmgmt(3N)`, `t_rcvconnect(3N)`, `t_snddis(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

WARNINGS

There may be transport provider-specific restrictions on address binding.

Some transport providers do not differentiate between a connection indication and the connection itself. If the connection has already been established after a successful return of `t_listen(3N)`, `t_accept()` will assign the existing connection to the transport endpoint specified by `resfd`.

| | |
|--------------------|--|
| NAME | t_alloc – allocate a library structure |
| SYNOPSIS | <pre>#include <xti.h> void *t_alloc(int fd, int struct_type, int fields);</pre> |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, <code>tiuser.h</code>, must be used. Refer to the section, <code>TLI COMPATIBILITY</code>, for a description of differences between the two interfaces.</p> <p>The <code>t_alloc()</code> function dynamically allocates memory for the various transport function argument structures as specified below. This function will allocate memory for the specified structure, and will also allocate memory for buffers referenced by the structure.</p> <p>The structure to allocate is specified by <code>struct_type</code> and must be one of the following:</p> <pre>T_BIND struct t_bind T_CALL struct t_call T_OPTMGMT struct t_optmgmt T_DIS struct t_discon T_UNITDATA struct t_unitdata T_UDEERROR struct t_uderr T_INFO struct t_info</pre> <p>where each of these structures may subsequently be used as an argument to one or more transport functions.</p> <p>Each of the above structures, except <code>T_INFO</code>, contains at least one field of type <code>struct netbuf</code>. For each field of this type, the user may specify that the buffer for that field should be allocated as well. The length of the buffer allocated will be equal to or greater than the appropriate size as returned in the <code>info</code> argument of <code>t_open(3N)</code> or <code>t_getinfo(3N)</code>. The relevant fields of the <code>info</code> argument are described in the following list. The <code>fields</code> argument specifies which buffers to allocate, where the argument is the bitwise-or of any of the following:</p> |

| | |
|---------|--|
| T_ADDR | The <i>addr</i> field of the <code>t_bind</code> , <code>t_call</code> , <code>t_unitdata</code> or <code>t_uderr</code> structures. |
| T_OPT | The <i>opt</i> field of the <code>t_optmgmt</code> , <code>t_call</code> , <code>t_unitdata</code> or <code>t_uderr</code> structures. |
| T_UDATA | The <i>udata</i> field of the <code>t_call</code> , <code>t_discon</code> or <code>t_unitdata</code> structures. |
| T_ALL | All relevant fields of the given structure. Fields which are not supported by the transport provider specified by <i>fd</i> will not be allocated. |

For each relevant field specified in *fields*, `t_alloc()` will allocate memory for the buffer associated with the field, and initialize the *len* field to zero and the *buf* pointer and *maxlen* field accordingly. Irrelevant or unknown values passed in fields are ignored. Since the length of the buffer allocated will be based on the same size information that is returned to the user on a call to `t_open(3N)` and `t_getinfo(3N)`, *fd* must refer to the transport endpoint through which the newly allocated structure will be passed. In the case where a `T_INFO` structure is to be allocated, *fd* may be set to any value. In this way the appropriate size information can be accessed. If the size value associated with any specified field is `T_INVALID`, `t_alloc()` will be unable to determine the size of the buffer to allocate and will fail, setting `t_errno` to `TSYSERR` and `errno` to `EINVAL`. See `t_open(3N)` or `t_getinfo(3N)`. If the size value associated with any specified field is `T_INFINITE`, then the behavior of `t_alloc()` is implementation-defined. For any field not specified in *fields*, *buf* will be set to the null pointer and *len* and *maxlen* will be set to zero. See `t_open(3N)` or `t_getinfo(3N)`.

The pointer returned if the allocation succeeds is suitably aligned so that it can be assigned to a pointer to any type of object and then used to access such an object or array of such objects in the space allocated.

Use of `t_alloc()` to allocate structures will help ensure the compatibility of user programs with future releases of the transport interface functions.

RETURN VALUES

On successful completion, `t_alloc()` returns a pointer to the newly allocated structure. On failure, a null pointer is returned.

VALID STATES

ALL - apart from `T_UNINIT`

ERRORS

On failure, `t_errno` is set to one of the following:

| | |
|-------|--|
| TBADF | <code>struct_type</code> is other than <code>T_INFO</code> and the specified file descriptor does not refer to a transport endpoint. |
|-------|--|

| | |
|---------------------------------|---|
| TNOSTRUCTYPE | Unsupported <i>struct_type</i> requested. This can include a request for a structure type which is inconsistent with the transport provider type specified, that is, connection-mode or connectionless-mode. |
| TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). |
| TSYSERR | A system error has occurred during execution of this function. |
| TLI COMPATIBILITY | The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below. |
| Interface Header | The XTI interfaces use the header file, <code>xti.h</code> . TLI interfaces should <i>not</i> use this header. They should use the header: <pre>#include <tiuser.h></pre> |
| Error Description Values | The <code>t_errno</code> values that can be set by the XTI interface and cannot be set by the TLI interface are: TPROTO TNOSTRUCTYPE |
| Special Buffer Sizes | Assume that the value associated with any field of <code>struct t_info</code> (argument returned by <code>t_open()</code> or <code>t_getinfo()</code>) that describes buffer limits is <code>-1</code> . Then the underlying service provider can support a buffer of unlimited size. If this is the case, <code>t_alloc()</code> will allocate a buffer with the default size 1024 bytes, which may be handled as described in the next paragraph. If the underlying service provider supports a buffer of unlimited size in the <code>netbuf</code> structure (see <code>t_connect(3N)</code>), <code>t_alloc()</code> will return a buffer of size 1024 bytes. If a larger size buffer is required, it will need to be allocated separately using a memory allocation routine such as <code>malloc(3C)</code> . The <code>buf</code> and <code>maxlen</code> fields of the <code>netbuf</code> data structure can then be updated with the address of the new buffer and the 1024 byte buffer originally allocated by <code>t_alloc()</code> can be freed using <code>free(3C)</code> . Assume that the value associated with any field of <code>struct t_info</code> (argument returned by <code>t_open()</code> or <code>t_getinfo()</code>) that describes <code>nbuffer</code> limits is <code>--2</code> . |

Then **t_alloc()** will set the buffer pointer to `NULL` and the buffer maximum size to 0, and then will return success (see **t_open(3N)** or **t_getinfo(3N)**).

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

free(3C), **malloc(3C)**, **t_connect(3N)**, **t_free(3N)**, **t_getinfo(3N)**, **t_open(3N)**, **attributes(5)**

NAME tan – tangent function

SYNOPSIS cc [*flag ...*] *file ...* -lm [*library ...*]
#include <math.h>
double **tan**(double *x*);

DESCRIPTION The **tan()** function computes the tangent of its argument *x*, measured in radians.

RETURN VALUES Upon successful completion, **tan()** returns the tangent of *x*.
If *x* is NaN or ±Inf, NaN is returned.

ERRORS No errors will occur.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **atan(3M)**, **isnan(3M)**, **attributes(5)**

NAME | tanh – hyperbolic tangent function

SYNOPSIS | cc [*flag ...*] *file ...* -lm [*library ...*]
| #include <math.h>
|
| double **tanh**(double *x*);

DESCRIPTION | The **tanh()** function computes the hyperbolic tangent of *x*.

RETURN VALUES | Upon successful completion, **tanh()** returns the hyperbolic tangent of *x*.
| If *x* is NaN, NaN is returned.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **atanh(3M)**, **isnan(3M)**, **tan(3M)**, **attributes(5)**

| | |
|--------------------|--|
| NAME | t_bind – bind an address to a transport endpoint |
| SYNOPSIS | <pre>#include <xti.h> int t_bind(int fd, const struct t_bind *req, struct t_bind *ret);</pre> |
| DESCRIPTION | <p>This routine is part of the XTI interfaces that evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the <code>TLI COMPATIBILITY</code> section for a description of differences between the two interfaces.</p> <p>This function associates a protocol address with the transport endpoint specified by <code>fd</code> and activates that transport endpoint. In connection mode, the transport provider may begin enqueueing incoming connect indications, or servicing a connection request on the transport endpoint. In connectionless-mode, the transport user may send or receive data units through the transport endpoint.</p> <p>The <code>req</code> and <code>ret</code> arguments point to a <code>t_bind</code> structure containing the following members:</p> <pre>struct netbuf addr; unsigned qlen;</pre> <p>The <code>addr</code> field of the <code>t_bind</code> structure specifies a protocol address, and the <code>qlen</code> field is used to indicate the maximum number of outstanding connection indications.</p> <p>The parameter <code>req</code> is used to request that an address, represented by the <code>netbuf</code> structure, be bound to the given transport endpoint. The parameter <code>len</code> specifies the number of bytes in the address, and <code>buf</code> points to the address buffer. The parameter <code>maxlen</code> has no meaning for the <code>req</code> argument. On return, <code>ret</code> contains an encoding for the address that the transport provider actually bound to the transport endpoint; if an address was specified in <code>req</code>, this will be an encoding of the same address. In <code>ret</code>, the user specifies <code>maxlen</code>, which is the maximum size of the address buffer, and <code>buf</code> which points to the buffer where the address is to be placed. On return, <code>len</code> specifies the number of bytes in the bound address, and <code>buf</code> points to the bound address. If <code>maxlen</code> equals zero, no address is returned. If <code>maxlen</code> is greater than zero and less than the length of the address, <code>t_bind()</code> fails with <code>t_errno</code> set to <code>TBUFOVFLW</code>.</p> |

If the requested address is not available, **t_bind()** will return -1 with `t_errno` set as appropriate. If no address is specified in *req* (the *len* field of *addr* in *req* is zero or *req* is NULL), the transport provider will assign an appropriate address to be bound, and will return that address in the *addr* field of *ret*. If the transport provider could not allocate an address, **t_bind()** will fail with `t_errno` set to TNOADDR.

The parameter *req* may be a null pointer if the user does not wish to specify an address to be bound. Here, the value of *qlen* is assumed to be zero, and the transport provider will assign an address to the transport endpoint. Similarly, *ret* may be a null pointer if the user does not care what address was bound by the provider and is not interested in the negotiated value of *qlen*. It is valid to set *req* and *ret* to the null pointer for the same call, in which case the provider chooses the address to bind to the transport endpoint and does not return that information to the user.

The *qlen* field has meaning only when initializing a connection-mode service. It specifies the number of outstanding connection indications that the transport provider should support for the given transport endpoint. An outstanding connection indication is one that has been passed to the transport user by the transport provider but which has not been accepted or rejected. A value of *qlen* greater than zero is only meaningful when issued by a passive transport user that expects other users to call it. The value of *qlen* will be negotiated by the transport provider and may be changed if the transport provider cannot support the specified number of outstanding connection indications. However, this value of *qlen* will never be negotiated from a requested value greater than zero to zero. This is a requirement on transport providers; see WARNINGS below. On return, the *qlen* field in *ret* will contain the negotiated value.

If *fd* refers to a connection-mode service, this function allows more than one transport endpoint to be bound to the same protocol address. but it is not possible to bind more than one protocol address to the same transport endpoint. However, the transport provider must also support this capability. If a user binds more than one transport endpoint to the same protocol address, only one endpoint can be used to listen for connection indications associated with that protocol address. In other words, only one **t_bind()** for a given protocol address may specify a value of *qlen* greater than zero. In this way, the transport provider can identify which transport endpoint should be notified of an incoming connection indication. If a user attempts to bind a protocol address to a second transport endpoint with a value of *qlen* greater than zero, **t_bind()** will return -1 and set `t_errno` to TADDRBUSY. When a user accepts a connection on the transport endpoint that is being used as the listening endpoint, the bound protocol address will be found to be busy for the duration of the connection, until a `t_unbind(3N)` or `t_close(3N)` call has been issued. No other transport endpoints may be bound for listening on that same protocol address while that initial listening endpoint is active (in the data transfer phase

or in the T_IDLE state). This will prevent more than one transport endpoint bound to the same protocol address from accepting connection indications.

If *fd* refers to connectionless mode service, this function allows for more than one transport endpoint to be associated with a protocol address, where the underlying transport provider supports this capability (often in conjunction with value of a protocol-specific option). If a user attempts to bind a second transport endpoint to an already bound protocol address when such capability is not supported for a transport provider, **t_bind()** will return -1 and set *t_errno* to TADDRBUSY.

RETURN VALUES

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *t_errno* is set to indicate an error.

VALID STATES

T_UNBND

ERRORS

On failure, *t_errno* is set to one of the following:

| | |
|-----------|---|
| TACCES | The user does not have permission to use the specified address. |
| TADDRBUSY | The requested address is in use. |
| TBADADDR | The specified protocol address was in an incorrect format or contained illegal information. |
| TBADF | The specified file descriptor does not refer to a transport endpoint. |
| TBUFOVFLW | The number of bytes allowed for an incoming argument (<i>maxlen</i>) is greater than 0 but not sufficient to store the value of that argument. The provider's state will change to T_IDLE and the information to be returned in <i>ret</i> will be discarded. |
| TOUTSTATE | The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. |
| TNOADDR | The transport provider could not allocate an address. |
| TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<i>t_errno</i>). |
| TSYSERR | A system error has occurred during execution of this function. |

**TLI
COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```
#include <tiuser.h>
```

Address Bound

The user can compare the addresses in *req* and *ret* to determine whether the transport provider bound the transport endpoint to a different address than that requested.

**Error Description
Values**

The `t_errno` values `TPROTO` and `TADDRBUSY` can be set by the XTI interface but cannot be set by the TLI interface.

A `t_errno` value that this routine can return under different circumstances than its XTI counterpart is `TBUFOVFLW`. It can be returned even when the `maxlen` field of the corresponding buffer has been set to zero.

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`t_accept(3N)`, `t_alloc(3N)`, `t_close(3N)`, `t_connect(3N)`,
`t_unbind(3N)`, `attributes(5)`

WARNINGS

The requirement that the value of *qlen* never be negotiated from a requested value greater than zero to zero implies that transport providers, rather than the XTI implementation itself, accept this restriction.

An implementation need not allow an application explicitly to bind more than one communications endpoint to a single protocol address, while permitting more than one connection to be accepted to the same protocol address. That means that although an attempt to bind a communications endpoint to some address with *qlen=0* might be rejected with `TADDRBUSY`, the user may nevertheless use this (unbound) endpoint as a responding endpoint in a call to `t_accept(3N)`. To become independent of such implementation differences, the user should supply unbound responding endpoints to `t_accept(3N)`.

The local address bound to an endpoint may change as result of a `t_accept(3N)` or `t_connect(3N)` call. Such changes are not necessarily reversed when the connection is released.

| NAME | tcdrain – wait for transmission of output | | | | |
|----------------------|---|----------------|-----------------|----------|--------------------------------|
| SYNOPSIS | #include <termios.h> int tcdrain(int <i>fildev</i>); | | | | |
| DESCRIPTION | The tcdrain() function waits until all output written to the object referred to by <i>fildev</i> is transmitted. The <i>fildev</i> argument is an open file descriptor associated with a terminal. Any attempts to use tcdrain() from a process which is a member of a background process group on a <i>fildev</i> associated with its controlling terminal, will cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is allowed to perform the operation, and no signal is sent. | | | | |
| RETURN VALUES | Upon successful completion, 0 is returned. Otherwise, -1 is returned and <i>errno</i> is set to indicate the error. | | | | |
| ERRORS | The tcdrain() function will fail if: EBADF The <i>fildev</i> argument is not a valid file descriptor. EINTR A signal interrupted tcdrain() . ENOTTY The file associated with <i>fildev</i> is not a terminal. The tcdrain() function may fail if: EIO The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: <table border="1" data-bbox="485 1339 1385 1430"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe, and Async-Signal-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe, and Async-Signal-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe, and Async-Signal-Safe | | | | |
| SEE ALSO | tcflush(3) , attributes(5) , termio(7I) | | | | |

| | |
|----------------------|---|
| NAME | tcfow – suspend or restart the transmission or reception of data |
| SYNOPSIS | #include <termios.h> int tcfow (int <i>filde</i> , int <i>action</i>); |
| DESCRIPTION | <p>The tcfow() function suspends transmission or reception of data on the object referred to by <i>filde</i>, depending on the value of <i>action</i>. The <i>filde</i> argument is an open file descriptor associated with a terminal.</p> <ul style="list-style-type: none"> ■ If <i>action</i> is TCOOFF, output is suspended. ■ If <i>action</i> is TCOON, suspended output is restarted. ■ If <i>action</i> is TCIOFF, the system transmits a STOP character, which is intended to cause the terminal device to stop transmitting data to the system. ■ If <i>action</i> is TCION, the system transmits a START character, which is intended to cause the terminal device to start transmitting data to the system. <p>The default on the opening of a terminal file is that neither its input nor its output are suspended.</p> <p>Attempts to use tcfow() from a process which is a member of a background process group on a <i>filde</i> associated with its controlling terminal, will cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is allowed to perform the operation, and no signal is sent.</p> |
| RETURN VALUES | Upon successful completion, 0 is returned. Otherwise, -1 is returned and <i>errno</i> is set to indicate the error. |
| ERRORS | <p>The tcfow() function will fail if:</p> <p>EBADF The <i>filde</i> argument is not a valid file descriptor.</p> <p>EINVAL The <i>action</i> argument is not a supported value.</p> <p>ENOTTY The file associated with <i>filde</i> is not a terminal.</p> <p>The tcfow() function may fail if:</p> <p>EIO The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.</p> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|--------------------------------|
| MT-Level | MT-Safe, and Async-Signal-Safe |

SEE ALSO`tcsendbreak(3)`, `attributes(5)`, `termio(7I)`

| NAME | tflush – flush non-transmitted output data, non-read input data or both | | | | |
|----------------------|---|----------------|-----------------|----------|--------------------------------|
| SYNOPSIS | #include <termios.h> int tflush (int <i>fildev</i> , int <i>queue_selector</i>); | | | | |
| DESCRIPTION | <p>Upon successful completion, tflush() discards data written to the object referred to by <i>fildev</i> (an open file descriptor associated with a terminal) but not transmitted, or data received but not read, depending on the value of <i>queue_selector</i>:</p> <ul style="list-style-type: none"> ■ If <i>queue_selector</i> is TCIFLUSH it flushes data received but not read. ■ If <i>queue_selector</i> is TCOFLUSH it flushes data written but not transmitted. ■ If <i>queue_selector</i> is TCIOFLUSH it flushes both data received but not read and data written but not transmitted. <p>Attempts to use tflush() from a process which is a member of a background process group on a <i>fildev</i> associated with its controlling terminal, will cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is allowed to perform the operation, and no signal is sent.</p> | | | | |
| RETURN VALUES | Upon successful completion, 0 is returned. Otherwise, -1 is returned and <i>errno</i> is set to indicate the error. | | | | |
| ERRORS | <p>The tflush() function will fail if:</p> <p>EBADF The <i>fildev</i> argument is not a valid file descriptor.</p> <p>EINVAL The <i>queue_selector</i> argument is not a supported value.</p> <p>ENOTTY The file associated with <i>fildev</i> is not a terminal.</p> <p>The tflush() function may fail if:</p> <p>EIO The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe, and Async-Signal-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe, and Async-Signal-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe, and Async-Signal-Safe | | | | |
| SEE ALSO | tcdrain(3) , attributes(5) , termio(7I) | | | | |

| NAME | tcgetattr – get the parameters associated with the terminal | | | | |
|----------------------|---|----------------|-----------------|----------|--------------------------------|
| SYNOPSIS | #include <termios.h> int tcgetattr(int <i>fildev</i> , struct termios * <i>termios_p</i>); | | | | |
| DESCRIPTION | The tcgetattr() function gets the parameters associated with the terminal referred to by <i>fildev</i> and stores them in the <code>termios</code> structure (see termio(7I)) referenced by <i>termios_p</i> . The <i>fildev</i> argument is an open file descriptor associated with a terminal. The <i>termios_p</i> argument is a pointer to a <code>termios</code> structure. The tcgetattr() operation is allowed from any process. If the terminal device supports different input and output baud rates, the baud rates stored in the <code>termios</code> structure returned by tcgetattr() reflect the actual baud rates, even if they are equal. If differing baud rates are not supported, the rate returned as the output baud rate is the actual baud rate. If the terminal device does not support split baud rates, the input baud rate stored in the <code>termios</code> structure will be 0. | | | | |
| RETURN VALUES | Upon successful completion, 0 is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error. | | | | |
| ERRORS | The tcgetattr() function will fail if: EBADF The <i>fildev</i> argument is not a valid file descriptor. ENOTTY The file associated with <i>fildev</i> is not a terminal. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: <table border="1" data-bbox="485 1348 1383 1438"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe, and Async-Signal-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe, and Async-Signal-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe, and Async-Signal-Safe | | | | |
| SEE ALSO | tcsetattr(3) , attributes(5) , termio(7I) | | | | |

| NAME | tcgetpgrp – get foreground process group ID | | | | |
|----------------------|---|----------------|-----------------|----------|--------------------------------|
| SYNOPSIS | <pre>#include <sys/types.h> #include <unistd.h> pid_t tcgetpgrp(int <i>fildev</i>);</pre> | | | | |
| DESCRIPTION | <p>The tcgetpgrp() function will return the value of the process group ID of the foreground process group associated with the terminal.</p> <p>If there is no foreground process group, tcgetpgrp() returns a value greater than 1 that does not match the process group ID of any existing process group.</p> <p>The tcgetpgrp() function is allowed from a process that is a member of a background process group; however, the information may be subsequently changed by a process that is a member of a foreground process group.</p> | | | | |
| RETURN VALUES | <p>Upon successful completion, tcgetpgrp() returns the value of the process group ID of the foreground process associated with the terminal. Otherwise, <code>-1</code> is returned and <code>errno</code> is set to indicate the error.</p> | | | | |
| ERRORS | <p>The tcgetpgrp() function will fail if:</p> <p>EBADF The <i>fildev</i> argument is not a valid file descriptor.</p> <p>ENOTTY The calling process does not have a controlling terminal, or the file is not the controlling terminal.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" data-bbox="500 1268 1396 1356"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe, and Async-Signal-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe, and Async-Signal-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe, and Async-Signal-Safe | | | | |
| SEE ALSO | setpgid(2) , setsid(2) , tcsetpgrp(3C) , attributes(5) , termio(7I) | | | | |

NAME tcgetsid – get process group ID for session leader for controlling terminal

SYNOPSIS #include <termios.h>

```
pid_t tcgetsid(int fildes);
```

DESCRIPTION The **tcgetsid()** function obtains the process group ID of the session for which the terminal specified by *fildes* is the controlling terminal.

RETURN VALUES Upon successful completion, **tcgetsid()** returns the process group ID associated with the terminal. Otherwise, a value of `(pid_t)-1` is returned and `errno` is set to indicate the error.

ERRORS The **tcgetsid()** function will fail if:

EACCES The *fildes* argument is not associated with a controlling terminal.

EBADF The *fildes* argument is not a valid file descriptor.

ENOTTY The file associated with *fildes* is not a terminal.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **attributes(5)**, **termio(7I)**

| | |
|----------------------|---|
| NAME | t_close – close a transport endpoint |
| SYNOPSIS | <pre>#include <xti.h> int t_close(int fd);</pre> |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>The <code>t_close()</code> function informs the transport provider that the user is finished with the transport endpoint specified by <code>fd</code>, and frees any local library resources associated with the endpoint. In addition, <code>t_close()</code> closes the file associated with the transport endpoint.</p> <p>The function <code>t_close()</code> should be called from the <code>T_UNBND</code> state. See <code>t_getstate(3N)</code>. However, this function does not check state information, so it may be called from any state to close a transport endpoint. If this occurs, the local library resources associated with the endpoint will be freed automatically. In addition, <code>close(2)</code> will be issued for that file descriptor; if there are no other descriptors in this process or in another process which references the communication endpoint, any connection that may be associated with that endpoint is broken. The connection may be terminated in an orderly or abortive manner.</p> <p>A <code>t_close()</code> issued on a connection endpoint may cause data previously sent, or data not yet received, to be lost. It is the responsibility of the transport user to ensure that data is received by the remote peer.</p> |
| RETURN VALUES | Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and <code>t_errno</code> is set to indicate an error. |
| VALID STATES | <code>T_UNBND</code> |
| ERRORS | <p>On failure, <code>t_errno</code> is set to the following:</p> <p><code>TBADF</code> The specified file descriptor does not refer to a transport endpoint.</p> <p><code>TPROTO</code> This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>).</p> |

TSYSERR A system error has occurred during execution of this function.

**TLI
COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```
#include <tiuser.h>
```

**Error Description
Values**

The `t_errno` value that can be set by the XTI interface and cannot be set by the TLI interface is:

TPROTO

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`close(2)`, `t_getstate(3N)`, `t_open(3N)`, `t_unbind(3N)`, `attributes(5)`
Transport Interfaces Programming Guide

| | |
|--------------------|---|
| NAME | t_connect – establish a connection with another transport user |
| SYNOPSIS | <pre>#include <xti.h></pre> <pre>int t_connect(int fd, const struct t_call *sndcall, struct t_call *rcvcall);</pre> |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces. This function enables a transport user to request a connection to the specified destination transport user.</p> <p>This function can only be issued in the <code>T_IDLE</code> state. The parameter <i>fd</i> identifies the local transport endpoint where communication will be established, while <i>sndcall</i> and <i>rcvcall</i> point to a <code>t_call</code> structure which contains the following members:</p> <pre>struct netbuf addr; struct netbuf opt; struct netbuf udata; int sequence;</pre> <p>The parameter <i>sndcall</i> specifies information needed by the transport provider to establish a connection and <i>rcvcall</i> specifies information that is associated with the newly established connection.</p> <p>In <i>sndcall</i>, <i>addr</i> specifies the protocol address of the destination transport user, <i>opt</i> presents any protocol-specific information that might be needed by the transport provider, <i>udata</i> points to optional user data that may be passed to the destination transport user during connection establishment, and <i>sequence</i> has no meaning for this function.</p> <p>On return, in <i>rcvcall</i>, <i>addr</i> contains the protocol address associated with the responding transport endpoint, <i>opt</i> represents any protocol-specific information associated with the connection, <i>udata</i> points to optional user data that may be returned by the destination transport user during connection establishment, and <i>sequence</i> has no meaning for this function.</p> <p>The <i>opt</i> argument permits users to define the options that may be passed to the transport provider. The user may choose not to negotiate protocol options</p> |

by setting the *len* field of *opt* to zero. In this case, the provider uses the option values currently set for the communications endpoint.

If used, *sndcall*→*opt.buf* must point to a buffer with the corresponding options, and *sndcall*→*opt.len* must specify its length. The *maxlen* and *buf* fields of the *netbuf* structure pointed by *rcvcall*→*addr* and *rcvcall*→*opt* must be set before the call.

The *udata* argument enables the caller to pass user data to the destination transport user and receive user data from the destination user during connection establishment. However, the amount of user data must not exceed the limits supported by the transport provider as returned in the *connect* field of the *info* argument of *t_open*(3N) or *t_getinfo*(3N). If the *len* of *udata* is zero in *sndcall*, no data will be sent to the destination transport user.

On return, the *addr*, *opt* and *udata* fields of *rcvcall* will be updated to reflect values associated with the connection. Thus, the *maxlen* field of each argument must be set before issuing this function to indicate the maximum size of the buffer for each. However, *maxlen* can be set to zero, in which case no information to this specific argument is given to the user on the return from *t_connect*(). If *maxlen* is greater than zero and less than the length of the value, *t_connect*() fails with *t_errno* set to *TBUFOVFLW*. If *rcvcall* is set to *NULL*, no information at all is returned.

By default, *t_connect*() executes in synchronous mode, and will wait for the destination user's response before returning control to the local user. A successful return (that is, return value of zero) indicates that the requested connection has been established. However, if *O_NONBLOCK* is set by means of *t_open*(3N) or *fcntl*(2), *t_connect*() executes in asynchronous mode. In this case, the call will not wait for the remote user's response, but will return control immediately to the local user and return -1 with *t_errno* set to *TNODATA* to indicate that the connection has not yet been established. In this way, the function simply initiates the connection establishment procedure by sending a connection request to the destination transport user. The *t_rcvconnect*(3N) function is used in conjunction with *t_connect*() to determine the status of the requested connection.

When a synchronous *t_connect*() call is interrupted by the arrival of a signal, the state of the corresponding transport endpoint is *T_OUTCON*, allowing a further call to either *t_rcvconnect*(3N), *t_rcvdis*(3N) or *t_snddis*(3N). When an asynchronous *t_connect*() call is interrupted by the arrival of a signal, the state of the corresponding transport endpoint is *T_IDLE*.

RETURN VALUES

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *t_errno* is set to indicate an error.

VALID STATES

T_IDLE

ERRORS

On failure, `t_errno` is set to one of the following:

| | |
|-------------|--|
| TACCES | The user does not have permission to use the specified address or options. |
| TADDRBUSY | This transport provider does not support multiple connections with the same local and remote addresses. This error indicates that a connection already exists. |
| TBADADDR | The specified protocol address was in an incorrect format or contained illegal information. |
| TBADDATA | The amount of user data specified was not within the bounds allowed by the transport provider. |
| TBADF | The specified file descriptor does not refer to a transport endpoint. |
| TBADOPT | The specified protocol options were in an incorrect format or contained illegal information. |
| TBUFOVFLW | The number of bytes allocated for an incoming argument (<i>maxlen</i>) is greater than 0 but not sufficient to store the value of that argument. If executed in synchronous mode, the provider's state, as seen by the user, changes to <code>T_DATAXFER</code> , and the information to be returned in <i>rcvcall</i> is discarded. |
| TLOOK | An asynchronous event has occurred on this transport endpoint and requires immediate attention. |
| TNODATA | <code>O_NONBLOCK</code> was set, so the function successfully initiated the connection establishment procedure, but did not wait for a response from the remote user. |
| TNOTSUPPORT | This function is not supported by the underlying transport provider. |
| TOUTSTATE | The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. |
| TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). |
| TSYSERR | A system error has occurred during execution of this function. |

**TLI
COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```
#include <tiuser.h>
```

**Error Description
Values**

The `TPROTO` and `TADDRBUSY` `t_errno` values can be set by the XTI interface but not by the TLI interface.

A `t_errno` value that this routine can return under different circumstances than its XTI counterpart is `TBUFOVFLW`. It can be returned even when the `maxlen` field of the corresponding buffer has been set to zero.

Option Buffers

The format of the options in an `opt` buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not fix the buffer format.

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`fcntl(2)`, `t_accept(3N)`, `t_alloc(3N)`, `t_getinfo(3N)`, `t_listen(3N)`, `t_open(3N)`, `t_optmgmt(3N)`, `t_rcvconnect(3N)`, `t_rcvdis(3N)`, `t_snddis(3N)`, `attributes`

Transport Interfaces Programming Guide

| NAME | tcsendbreak – send a “break” for a specific duration | | | | |
|----------------------|---|----------------|-----------------|----------|--------------------------------|
| SYNOPSIS | #include <termios.h> int tcsendbreak (int <i>fdes</i> , int <i>duration</i>); | | | | |
| DESCRIPTION | The <i>fdes</i> argument is an open file descriptor associated with a terminal. If the terminal is using asynchronous serial data transmission, tcsendbreak() will cause transmission of a continuous stream of zero-valued bits for a specific duration. If <i>duration</i> is 0, it will cause transmission of zero-valued bits for at least 0.25 seconds, and not more than 0.5 seconds. If <i>duration</i> is not 0, it behaves in a way similar to tcdrain(3) . If the terminal is not using asynchronous serial data transmission, it sends data to generate a break condition or returns without taking any action. Attempts to use tcsendbreak() from a process which is a member of a background process group on a <i>fdes</i> associated with its controlling terminal will cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is allowed to perform the operation, and no signal is sent. | | | | |
| RETURN VALUES | Upon successful completion, 0 is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error. | | | | |
| ERRORS | The tcsendbreak() function will fail if: EBADF The <i>fdes</i> argument is not a valid file descriptor. ENOTTY The file associated with <i>fdes</i> is not a terminal. The tcsendbreak() function may fail if: EIO The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe, and Async-Signal-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe, and Async-Signal-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe, and Async-Signal-Safe | | | | |
| SEE ALSO | tcdrain(3) , attributes(5) , termio(7I) | | | | |

| | |
|--------------------|---|
| NAME | tcsetattr – set the parameters associated with the terminal |
| SYNOPSIS | <pre>#include <termios.h> int tcsetattr(int <i>fildev</i>, int <i>optional_actions</i>, const struct termios *<i>termios_p</i>);</pre> |
| DESCRIPTION | <p>The tcsetattr() function sets the parameters associated with the terminal referred to by the open file descriptor <i>fildev</i> (an open file descriptor associated with a terminal) from the <code>termios</code> structure (see termio(7I)) referenced by <i>termios_p</i> as follows:</p> <ul style="list-style-type: none"> ■ If <i>optional_actions</i> is <code>TCSANOW</code>, the change will occur immediately. ■ If <i>optional_actions</i> is <code>TCSADRAIN</code>, the change will occur after all output written to <i>fildev</i> is transmitted. This function should be used when changing parameters that affect output. ■ If <i>optional_actions</i> is <code>TCSAFLUSH</code>, the change will occur after all output written to <i>fildev</i> is transmitted, and all input so far received but not read will be discarded before the change is made. <p>If the output baud rate stored in the <code>termios</code> structure pointed to by <i>termios_p</i> is the zero baud rate, <code>B0</code>, the modem control lines will no longer be asserted. Normally, this will disconnect the line.</p> <p>If the input baud rate stored in the <code>termios</code> structure pointed to by <i>termios_p</i> is 0, the input baud rate given to the hardware will be the same as the output baud rate stored in the <code>termios</code> structure.</p> <p>The tcsetattr() function will return successfully if it was able to perform any of the requested actions, even if some of the requested actions could not be performed. It will set all the attributes that implementation supports as requested and leave all the attributes not supported by the implementation unchanged. If no part of the request can be honoured, it will return <code>-1</code> and set <code>errno</code> to <code>EINVAL</code>. If the input and output baud rates differ and are a combination that is not supported, neither baud rate is changed. A subsequent call to tcgetattr(3) will return the actual state of the terminal device (reflecting both the changes made and not made in the previous tcsetattr() call). The tcsetattr() function will not change the values in the <code>termios</code> structure whether or not it actually accepts them.</p> <p>The effect of tcsetattr() is undefined if the value of the <code>termios</code> structure pointed to by <i>termios_p</i> was not derived from the result of a call to tcgetattr(3) on <i>fildev</i>; an application should modify only fields and flags defined by this document between the call to tcgetattr(3) and tcsetattr(), leaving all other fields and flags unmodified.</p> |

No actions defined by this document, other than a call to **tcsetattr()** or a close of the last file descriptor in the system associated with this terminal device, will cause any of the terminal attributes defined by this document to change.

Attempts to use **tcsetattr()** from a process which is a member of a background process group on a *fildev* associated with its controlling terminal, will cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is allowed to perform the operation, and no signal is sent.

USAGE

If trying to change baud rates, applications should call **tcsetattr()** then call **tcgetattr(3)** in order to determine what baud rates were actually selected.

RETURN VALUES

Upon successful completion, 0 is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.

ERRORS

The **tcsetattr()** function will fail if:

EBADF The *fildev* argument is not a valid file descriptor.

EINTR A signal interrupted **tcsetattr()**.

EINVAL The *optional_actions* argument is not a supported value, or an attempt was made to change an attribute represented in the `termios` structure to an unsupported value.

ENOTTY The file associated with *fildev* is not a terminal.

The **tcsetattr()** function may fail if:

EIO The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|--------------------------------|
| MT-Level | MT-Safe, and Async-Signal-Safe |

SEE ALSO

cfgetispeed(3), **tcgetattr(3)**, **attributes(5)**, **termio(7I)**

| NAME | tcsetpgrp – set foreground process group ID | | | | |
|----------------------|--|----------------|-----------------|----------|--------------------------------|
| SYNOPSIS | <pre>#include <sys/types.h> #include <unistd.h> int tcsetpgrp(int <i>fildev</i>, pid_t <i>pgid_id</i>);</pre> | | | | |
| DESCRIPTION | <p>If the process has a controlling terminal, tcsetpgrp() will set the foreground process group ID associated with the terminal to <i>pgid_id</i>. The file associated with <i>fildev</i> must be the controlling terminal of the calling process and the controlling terminal must be currently associated with the session of the calling process. The value of <i>pgid_id</i> must match a process group ID of a process in the same session as the calling process.</p> | | | | |
| RETURN VALUES | <p>Upon successful completion, 0 is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error.</p> | | | | |
| ERRORS | <p>The tcsetpgrp() function will fail if:</p> <p>EBADF The <i>fildev</i> argument is not a valid file descriptor.</p> <p>EINVAL This implementation does not support the value in the <i>pgid_id</i> argument.</p> <p>ENOTTY The calling process does not have a controlling terminal, or the file is not the controlling terminal, or the controlling terminal is no longer associated with the session of the calling process.</p> <p>EPERM The value of <i>pgid_id</i> does not match the process group ID of a process in the same session as the calling process.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe, and Async-Signal-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe, and Async-Signal-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe, and Async-Signal-Safe | | | | |
| SEE ALSO | <p>tcgetpgrp(3), attributes(5), termio(7I)</p> | | | | |

| NAME | tcsetpgrp – set foreground process group ID of terminal | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <unistd.h> int tcsetpgrp(int <i>fildev</i>, pid_t <i>pgid</i>);</pre> | | | | |
| DESCRIPTION | The tcsetpgrp() function sets the foreground process group ID of the terminal specified by <i>fildev</i> to <i>pgid</i> . The file associated with <i>fildev</i> must be the controlling terminal of the calling process and the controlling terminal must be currently associated with the session of the calling process. The value of <i>pgid</i> must match a process group ID of a process in the same session as the calling process. | | | | |
| RETURN VALUES | Upon successful completion, tcsetpgrp() returns 0. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error. | | | | |
| ERRORS | <p>The tcsetpgrp() function will fail if:</p> <p>EBADF The <i>fildev</i> argument is not a valid file descriptor.</p> <p>EINVAL The <i>fildev</i> argument is a terminal that does not support tcsetpgrp(), or <i>pgid</i> is not a valid process group ID.</p> <p>EIO The process is not ignoring or holding SIGTTOU and is a member of an orphaned process group.</p> <p>ENOTTY The calling process does not have a controlling terminal, or the file is not the controlling terminal, or the controlling terminal is no longer associated with the session of the calling process.</p> <p>EPERM The value of <i>pgid</i> does not match the process group ID of an existing process in the same session as the calling process.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | attributes(5) , termio(7I) | | | | |

NAME td_init – performs initialization for libthread_db library of interfaces

SYNOPSIS cc [*flag...*] *file...*/lib/libthread_db.so.1 [*library...*]

```
#include <proc_service.h>
#include <thread_db.h>

td_err_e td_init();
```

DESCRIPTION **td_init()** is the global initialization function for the **libthread_db()** library of interfaces. It must be called exactly once by any process using the **libthread_db()** library before any other libthread_db function can be called.

RETURN VALUES

TD_OK The **libthread_db()** library of interfaces successfully initialized.

TD_ERR Initialization failed.

ATTRIBUTES See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO **libthread_db(3T)**, **libthread_db(4)**, **attributes(5)**

NAME td_log – placeholder for future logging functionality

SYNOPSIS cc [*flag...*] *file...*/lib/libthread_db.so.1 [*library...*]

```
#include <proc_service.h>
#include <thread_db.h>
```

```
void td_log();
```

DESCRIPTION This function presently does nothing; it is merely a placeholder for future logging functionality in `libthread_db(3T)`.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO `libthread(3T)`, `libthread_db(3T)`, `libthread_db(4)`, `attributes(5)`

| | | | |
|--------------------------------------|---|--------------------------------------|--|
| NAME | td_sync_get_info, td_sync_setstate, td_sync_waiters – operations on a synchronization object in libthread_db | | |
| SYNOPSIS | <pre> cc [flag ...] file .../lib/libthread_db.so.1 [library ...] #include <proc_service.h> #include <thread_db.h> td_err_e td_sync_get_info(const td_synchandle_t * sh_p, td_syncinfo_t * si_p); td_err_e td_sync_setstate(const td_synchandle_t * sh_p); td_err_e td_sync_waiters(const td_synchandle_t * sh_p, td_thr_iter_f * cb, void * cb_data_p); </pre> | | |
| DESCRIPTION | <p>Synchronization objects include mutexes, condition variables, semaphores, and reader-writer locks. In the same way that thread operations use a thread handle of type <code>td_thrhandle_t</code>, operations on synchronization objects use a synchronization object handle of type <code>td_synchandle_t</code>.</p> <p>The controlling process obtains synchronization object handles either by calling the function <code>td_ta_sync_iter()</code> to obtain handles for all synchronization objects of the target process that are known to the <code>libthread_db</code> library of interfaces, or by mapping the address of a synchronization object in the address space of the target process to a handle by calling <code>td_ta_map_addr2sync()</code>.</p> <p>Note that not all synchronization objects that a process uses may be known to the <code>libthread_db</code> library and returned by <code>td_ta_sync_iter()</code>. A synchronization object is known to <code>libthread_db</code> only if it was ever waited on after <code>libthread_db</code> was attached to the process. For example, a mutex may have been widely used, but if no thread ever blocked waiting to acquire it, it will not be known to <code>libthread_db</code> interfaces.</p> <p>The <code>td_sync_get_info()</code> function fills in the <code>td_syncinfo_t</code> structure <code>* si_p</code> with values for the synchronization object identified by <code>sh_p</code>. The <code>td_syncinfo_t</code> structure contains the following fields:</p> <table border="0" style="width: 100%;"> <tr> <td style="padding-right: 20px;"><code>td_thragent_t * si_ta_p</code></td> <td>The internal process handle identifying the target process through which this synchronization object handle was obtained. Synchronization objects may be process-private or process-shared. In</td> </tr> </table> | <code>td_thragent_t * si_ta_p</code> | The internal process handle identifying the target process through which this synchronization object handle was obtained. Synchronization objects may be process-private or process-shared. In |
| <code>td_thragent_t * si_ta_p</code> | The internal process handle identifying the target process through which this synchronization object handle was obtained. Synchronization objects may be process-private or process-shared. In | | |

| | | |
|-----------------|-----------------------|---|
| | | the latter case, the same synchronization object may have multiple handles, one for each target process's "view" of the synchronization object. |
| psaddr_t | si_sv_addr | The address of the synchronization object in this target process's address space. |
| td_sync_type_e | si_type | The type of the synchronization variable: mutex, condition variable, semaphore, or reader-writer lock. |
| int | si_shared_type | USYNC_THREAD if this synchronization object is process-private; USYNC_PROCESS if it is process-shared. |
| td_sync_flags_t | si_flags | Flags dependent on the type of the synchronization object. |
| int | si_state.sema_count | Semaphores only. The current value of the semaphore |
| int | si_state.nreaders | Reader-writer locks only. The number of readers currently holding the lock, or -1 , if a writer is currently holding the lock. |
| int | si_state.mutex_locked | For mutexes only. Non-zero if and only if the mutex is currently locked. |
| int | si_size | The size of the synchronization object. |
| uchar_t | si_has_waiters | Non-zero if and only if at least one thread is blocked on this synchronization object. |
| uchar_t | si_is_wlocked | For reader-writer locks only. The value is non-zero if and only if this lock is held by a writer. |
| td_thrhandle_t | si_owner | Mutexes and reader-writer locks only. This is the thread holding the mutex, or the write lock, if this is a reader-writer lock. The value is NULL |

if no one holds the mutex or write-lock.

psaddr_t si_data

A pointer to optional data associated with the synchronization object. Currently useful only for debugging **libthread()** interfaces.

`td_sync_setstate` modifies the state of synchronization object `si_p`, depending on the synchronization object type. For mutexes, `td_sync_setstate` is unlocked if the value is 0. Otherwise it is locked. For semaphores, the semaphore's count is set to the value. For reader-writer locks, the reader count set to the value if value is >0. The count is set to write-locked if value is --1. It is set to unlocked if the value is 0. Setting the state of a synchronization object from a `libthread_db` interface may cause the synchronization object's semantics to be violated from the point of view of the threads in the target process. For example, if a thread holds a mutex, and `td_sync_setstate` is used to set the mutex to unlocked, then a different thread will also be able to subsequently acquire the same mutex.

`td_sync_waiters` iterates over the set of thread handles of threads blocked on `sh_p`. The callback function `cb` is called once for each such thread handle, and is passed the thread handle and `cb_data_p`. If the callback function returns a non-zero value, iteration is terminated early. See also `td_ta_thr_iter(3T)`.

RETURN VALUES

TD_OK The call returned successfully.

TD_BADTH An invalid thread handle was passed in.

TD_DBERR A call to one of the imported interface routines failed.

TD_ERR A libthread_db-internal error occurred.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`libthread_db(3T)`, `td_ta_map_addr2sync(3T)`, `td_ta_sync_iter(3T)`, `td_ta_thr_iter(3T)`, `libthread_db(4)`, `attributes(5)`

| | |
|--------------------|--|
| NAME | td_ta_enable_stats, td_ta_reset_stats, td_ta_get_stats – collect target process statistics for libthread_db |
| SYNOPSIS | <pre>cc [flag ...] file .../lib/libthread_db.so.1 [library ...] #include <proc_service.h> #include <thread_db.h> td_err_e td_ta_enable_stats(const td_thragent_t * ta_p, int on_off); td_err_e td_ta_reset_stats(const td_thragent_t * ta_p); td_err_e td_ta_get_stats(const td_thragent_t * ta_p, td_ta_stats_t * tstats);</pre> |
| DESCRIPTION | <p>The controlling process may request the collection of certain statistics about a target process. Statistics gathering is disabled by default; however, each target process has a <code>td_ta_stats_t</code> structure that contains up to date values when statistic gathering is enabled. td_ta_enable_stats() turns statistics gathering on or off for the process identified by <code>ta_p</code> depending on whether or not <code>on_off</code> is non-zero. When statistics gathering is turned on, all statistics are implicitly reset as though td_ta_reset_stats() had been called. Statistics are not reset when statistics gathering is turned off. Except for <code>nthreads</code> and <code>r_concurrency</code>, the values do not change further, but they remain available for inspection by way of td_ta_get_stats(). td_ta_reset_stats() resets all counters in the <code>td_ta_stats_t</code> structure to zero for the target process. td_ta_get_stats() returns the <code>td_ta_stats_t</code> structure for the process in <code>* stats_t</code>. The <code>td_ta_stats_t</code> structure is defined as follows:</p> <pre>typedef struct { int nthreads ;\011\011\011 /* total number of threads in use */ int r_concurrency ;\011\011 /* requested concurrency level */ int nrunnable_num ;\011\011 /* numerator of avg. runnable threads */ int nrunnable_den ;\011\011 /* denominator of avg. runnable threads */ int a_concurrency_num ; /* numerator, avg. achieved concurrency */ int</pre> |

```

a_concurrency_den
; /* denominator, avg. achieved concurrency */
int
nlwps_num
;\011\011 /* numerator, average number of LWPs in use */
int
nlwps_den
;\011\011 /* denominator, avg. number of LWPs in use */
int
nidle_num
;\011\011 /* numerator, avg. number of idling LWPs */
int
nidle_den
;\011\011 /* denominator, avg. number of idling LWPs */
} td_ta_stats_t;

```

nthreads is the number of threads that are currently part of the target process. *r_concurrency* is the current requested concurrency level, such as would be returned by `thr_setconcurrency(3T)`. The remaining fields are averages over time, each expressed as a fraction with an integral numerator and denominator. *nrunnable* is the average number of runnable threads. *a_concurrency* is the average achieved concurrency, the number of actually running threads. *a_concurrency* is less than or equal to *nrunnable*. *nlwps* is the average number of lightweight processes (LWP s) participating in this process. It must be greater than or equal to *a_concurrency*, as every running thread is assigned to an LWP, but there may at times be additional idling LWP s with no thread assigned to them. *nidle* is the average number of idle LWP s.

RETURN VALUES

TD_OK The call completed successfully.

TD_BADTA An invalid internal process handle was passed in.

TD_DBERR A call to one of the imported interface routines failed.

TD_ERR Something else went wrong.

ATTRIBUTES

See `attributes(5)` for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`libthread_db(3T)`, `thr_getconcurrency(3T)`, `libthread_db(4)`, `attributes(5)`

| | |
|--------------------|--|
| NAME | td_ta_event_addr, td_thr_event_enable, td_ta_set_event, td_thr_set_event, td_ta_clear_event, td_thr_clear_event, td_ta_event_getmsg, td_thr_event_getmsg, td_event_emptyset, td_event_fillset, td_event_addset, td_event_delset, td_eventismember, td_eventisempty – thread events in libthread_db |
| SYNOPSIS | <pre> cc [flag ...] file .../lib/libthread_db.so.1 [library ...] #include <proc_service.h> #include <thread_db.h> td_err_e td_ta_event_addr(const td_thragent_t * ta_p, u_long event, td_notify_t * notify_p); td_err_e td_thr_event_enable(const td_thrhandle_t * th_p, int on_off); td_err_e td_thr_set_event(const td_thrhandle_t * th_p, td_thr_events_t * events); td_err_e td_ta_set_event(const td_thragent_t * ta_p, td_thr_events_t * events); td_err_e td_thr_clear_event(const td_thrhandle_t * th_p, td_thr_events_t * events); td_err_e td_ta_clear_event(const td_thragent_t * ta_p, td_thr_events_t * events); td_err_e td_thr_event_getmsg(const td_thrhandle_t * th_p, td_event_msg_t * msg); td_err_e td_ta_event_getmsg(const td_thragent_t * ta_p, td_event_msg_t * msg); void td_event_emptyset(td_thr_events_t *); void td_event_fillset(td_thr_events_t *); void td_event_addset(td_thr_events_t *, td_thr_events_e n); void td_event_delset(td_thr_events_t *, td_thr_events_e n); void td_eventismember(td_thr_events_t *, td_thr_events_e n); void td_eventisempty(td_thr_events_t *); </pre> |
| DESCRIPTION | <p>These routines comprise the thread event facility for <code>libthread_db(3T)</code>. This facility allows the controlling process to be notified when certain thread-related events occur in a target process and to retrieve information associated with these events. An event consists of an event type, and</p> |

optionally, some associated event data, depending on the event type. See the section titled "Event Set Manipulation Macros" that follows.

The event type and the associated event data, if any, constitute an "event message." "Reporting an event" means delivering an event message to the controlling process by way of `libthread_db`.

Several flags can control event reporting, both a per-thread and per event basis. Event reporting may further be enabled or disabled for a thread. There is not only a per-thread event mask that specifies which event types should be reported for that thread, but there is also a global event mask that applies to all threads.

An event is reported, if and only if, the executing thread has event reporting enabled, and either the event type is enabled in the executing thread's event mask, or the event type is enabled in the global event mask.

Each thread has associated with it an event buffer in which it stores the most recent event message it has generated, the type of the most recent event that it reported, and, depending on the event type, some additional information related to that event. See the section titled "Event Set Manipulation Macros" for a description of the `td_thr_events_e` and `td_event_msg_t` types and a list of the event types and the values reported with them. The thread handle, type `td_thrhandle_t`, the event type, and the possible value, together constitute an event message. Each thread's event buffer holds at most one event message.

Each event type has an event reporting address associated with it. A thread reports an event by writing the event message into the thread's event buffer and having control reach the event reporting address for that event type.

Typically, the controlling process sets a breakpoint at the event reporting address for one or more event types. When the breakpoint is hit, the controlling process knows that an event of the corresponding type has occurred.

The event types, and the additional information, if any, reported with each event, are:

| | |
|----------------------------|---|
| <code>TD_READY</code> | The thread became ready to execute. |
| <code>TD_SLEEP</code> | The thread has blocked on a synchronization object. |
| <code>TD_SWITCHTO</code> | A runnable thread is being assigned to LWP. |
| <code>TD_SWITCHFROM</code> | A running thread is being removed from its LWP. |
| <code>TD_LOCK_TRY</code> | A thread is trying to get an unavailable lock. |
| <code>TD_CATCHSIG</code> | A signal was posted to a thread. |

| | |
|----------------|--|
| TD_IDLE | An LWP is becoming idle. |
| TD_CREATE | A thread is being created. |
| TD_DEATH | A thread has terminated. |
| TD_PREEMPT | A thread is being preempted. |
| TD_PRI_INHERIT | A thread is inheriting an elevated priority from another thread. |
| TD_REAP | A thread is being reaped. |
| TD_CONCURRENCY | The number of LWPs is changing. |

TD_TIMEOUT A condition-variable timed wait expired.

td_ta_event_addr() returns in ** notify_p* the event reporting address associated with event type *event* . The controlling process may then set a breakpoint at that address. If a thread hits that breakpoint, it reports an event of type *event* .

td_thr_event_enable() enables or disables event reporting for thread *th_p* . If a thread has event reporting disabled, it will not report any events. Threads are started with event reporting disabled. Event reporting is enabled if *on_off* is non-zero; otherwise, it is disabled. To find out whether or not event reporting is enabled on a thread, call **td_thr_getinfo()** for the thread and examine the *ti_traceme* field of the *td_thrinfo_t* structure it returns.

td_thr_set_event() and **td_thr_clear_event()** set and clear, respectively, a set of event types in the event mask associated with the thread *th_p* . To inspect a thread's event mask, call **td_thr_getinfo()** for the thread, and examine the *ti_events* field of the *td_thrinfo_t* structure it returns.

td_ta_set_event() and **td_ta_clear_event()** are just like **td_thr_set_event()** and **td_thr_clear_event()** , respectively, except that the target process's global event mask is modified. There is no provision for inspecting the value of a target process's global event mask.

td_thr_event_getmsg() returns in ** msg* the event message associated with thread ** th_p* . Reading a thread's event message consumes the message, emptying the thread's event buffer. As noted above, each thread's event buffer holds at most one event message; if a thread reports a second event before the first event message has been read, the second event message overwrites the first.

td_ta_event_getmsg() is just like **td_thr_event_getmsg()** , except that it is passed a process handle rather than a thread handle. It selects some thread that

has an event message buffered, and it returns that thread's message. The thread selected is undefined, except that as long as at least one thread has an event message buffered, it will return an event message from some such thread.

Event Set Manipulation Macros

Several macros are provided for manipulating event sets of type

`td_thr_events_t` :

| | |
|--------------------------------|--|
| <code>td_event_emptyset</code> | Sets its argument to the <code>NULL</code> event set. |
| <code>td_event_fillset</code> | Sets its argument to the set of all events. |
| <code>td_event_addset</code> | Adds a specific event type to an event set. |
| <code>td_event_delset</code> | Deletes a specific event type from an event set. |
| <code>td_eventismember</code> | Tests whether a specific event type is a member of an event set. |
| <code>td_eventisempty</code> | Tests whether an event set is the <code>NULL</code> set. |

RETURN VALUES

The following values may be returned for all thread event routines:

| | |
|-----------------------|---|
| <code>TD_OK</code> | The call returned successfully. |
| <code>TD_BADTH</code> | An invalid thread handle was passed in. |
| <code>TD_BADTA</code> | An invalid internal process handle was passed in. |
| <code>TD_BADPH</code> | There is a <code>NULL</code> external process handle associated with this internal process handle. |
| <code>TD_DBERR</code> | A call to one of the imported interface routines failed. |
| <code>TD_NOMSG</code> | No event message was available to return to <code>td_thr_event_getmsg()</code> or <code>td_ta_event_getmsg()</code> . |
| <code>TD_ERR</code> | Some other parameter error occurred, or a <code>libthread_db()</code> internal error occurred. |

The following value may be returned for `td_thr_event_enable()` , `td_thr_set_event()` , and `td_thr_clear_event()` only:

| | |
|-------------------------|--|
| <code>TD_NOCAPAB</code> | The agent thread in the target process has not completed initialization, so this operation cannot be performed. The operation can be performed after the target process has been allowed to make some forward progress. See also <code>libthread_db(3T)</code> . |
|-------------------------|--|

ATTRIBUTES

See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

libthread_db(3T) , **libthread_db(4)** , **attributes(5)**

NAME | `td_ta_get_nthreads` – gets the total number of threads in a process for `libthread_db`

SYNOPSIS | `cc [flag...] file.../lib/libthread_db.so.1 [library...]`

```
#include <proc_service.h>
#include <thread_db.h>

td_err_e td_ta_get_nthreads(const td_thragent_t *ta_p, int *nthread_p);
```

DESCRIPTION | `td_ta_get_nthreads()` returns the total number of threads in process *ta_p*, including any system threads. System threads are those created by `libthread()` or `libthread_db()` on its own behalf. The number of threads is written into *nthread_p*.

RETURN VALUES

| | |
|----------|---|
| TD_OK | The call completed successfully. |
| TD_BADTA | An invalid internal process handle was passed in. |
| TD_BADPH | There is a NULL external process handle associated with this internal process handle. |
| TD_DBERR | A call to one of the imported interface routines failed. |
| TD_ERR | <i>nthread_p</i> was NULL, or a <code>libthread_db</code> internal error occurred. |

ATTRIBUTES | See `attributes(5)` for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO | `libthread(3T)`, `libthread_db(3T)`, `libthread_db(4)`, `attributes(5)`

| NAME | td_ta_map_addr2sync – get a synchronization object handle from a synchronization object's address | | | | | | | | | | | | |
|----------------------|--|----------------|----------------------------------|----------|---|----------|---|----------|---|----------|--|--------|--|
| SYNOPSIS | <pre>cc [<i>flag...</i>] <i>file...</i>/lib/libthread_db.so.1 [<i>library...</i>] #include <proc_service.h> #include <thread_db.h> td_ta_map_addr2sync(const td_thragent_t *ta_p, psaddr_t addr,td_synchronize_t *sh_p);</pre> | | | | | | | | | | | | |
| DESCRIPTION | td_ta_map_addr2sync() produces the synchronization object handle of type <code>td_synchronize_t</code> that corresponds to the address of the synchronization object (mutex, semaphore, condition variable, or reader/writer lock). Some effort is made to validate <code>addr</code> and verify that it does indeed point at a synchronization object. The handle is returned in <code>*sh_p</code> . | | | | | | | | | | | | |
| RETURN VALUES | <table border="0"> <tr> <td style="padding-right: 20px;">TD_OK</td> <td>The call completed successfully.</td> </tr> <tr> <td>TD_BADTA</td> <td>An invalid internal process handle was passed in.</td> </tr> <tr> <td>TD_BADPH</td> <td>There is a NULL external process handle associated with this internal process handle.</td> </tr> <tr> <td>TD_BADSH</td> <td><code>sh_p</code> is NULL, or <code>addr</code> does not appear to point to a valid synchronization object.</td> </tr> <tr> <td>TD_DBERR</td> <td>A call to one of the imported interface routines failed.</td> </tr> <tr> <td>TD_ERR</td> <td><code>addr</code> is NULL, or a <code>libthread_db</code> internal error occurred.</td> </tr> </table> | TD_OK | The call completed successfully. | TD_BADTA | An invalid internal process handle was passed in. | TD_BADPH | There is a NULL external process handle associated with this internal process handle. | TD_BADSH | <code>sh_p</code> is NULL, or <code>addr</code> does not appear to point to a valid synchronization object. | TD_DBERR | A call to one of the imported interface routines failed. | TD_ERR | <code>addr</code> is NULL, or a <code>libthread_db</code> internal error occurred. |
| TD_OK | The call completed successfully. | | | | | | | | | | | | |
| TD_BADTA | An invalid internal process handle was passed in. | | | | | | | | | | | | |
| TD_BADPH | There is a NULL external process handle associated with this internal process handle. | | | | | | | | | | | | |
| TD_BADSH | <code>sh_p</code> is NULL, or <code>addr</code> does not appear to point to a valid synchronization object. | | | | | | | | | | | | |
| TD_DBERR | A call to one of the imported interface routines failed. | | | | | | | | | | | | |
| TD_ERR | <code>addr</code> is NULL, or a <code>libthread_db</code> internal error occurred. | | | | | | | | | | | | |
| ATTRIBUTES | See attributes(5) for description of the following attributes: <table border="1" style="margin-top: 10px; width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Safe | | | | | | | | |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | | | | | | | |
| MT-Level | Safe | | | | | | | | | | | | |
| SEE ALSO | libthread_db(3T) , libthread_db(4) , attributes(5) | | | | | | | | | | | | |

| | | | | | | | | | | | | | |
|----------------------|--|-------|----------------------------------|----------|---|----------|---|----------|--|----------|--|--------|---|
| NAME | td_ta_map_id2thr, td_ta_map_lwp2thr – convert a thread id or LWP id to a thread handle | | | | | | | | | | | | |
| SYNOPSIS | <pre>cc [flag ...] file .../lib/libthread_db.so.1 [library ...] #include <proc_service.h> #include <thread_db.h> td_ta_map_id2thr(const td_thragent_t * ta_p, thread_t tid,td_thrhandle_t * th_p); td_ta_map_lwp2thr(const td_thragent_t * ta_p, lwpid_t lwpid,td_thrhandle_t * th_p);</pre> | | | | | | | | | | | | |
| DESCRIPTION | <p>td_ta_map_id2thr() produces the <code>td_thrhandle_t</code> thread handle that corresponds to a particular thread id, as returned by <code>thr_create(3T)</code> or <code>thr_self(3T)</code>. The thread handle is returned in <code>* th_p</code>.</p> <p>td_ta_map_lwp2thr() produces the <code>td_thrhandle_t</code> thread handle for the thread that is currently executing on the light weight process (LWP)and has an id of <code>lwpid</code>.</p> | | | | | | | | | | | | |
| RETURN VALUES | <table border="0"> <tr> <td style="padding-right: 20px;">TD_OK</td> <td>The call completed successfully.</td> </tr> <tr> <td>TD_BADTA</td> <td>An invalid internal process handle was passed in.</td> </tr> <tr> <td>TD_BADPH</td> <td>There is a NULL external process handle associated with this internal process handle.</td> </tr> <tr> <td>TD_DBERR</td> <td>A call to one of the imported interface routines failed.</td> </tr> <tr> <td>TD_NOTHR</td> <td>Either there is no thread with the given thread id (<code>td_ta_map_id2thr</code>)or no thread is currently executing on the given LWP (<code>td_ta_map_lwp2thr</code>).</td> </tr> <tr> <td>TD_ERR</td> <td>The call did not complete successfully.</td> </tr> </table> | TD_OK | The call completed successfully. | TD_BADTA | An invalid internal process handle was passed in. | TD_BADPH | There is a NULL external process handle associated with this internal process handle. | TD_DBERR | A call to one of the imported interface routines failed. | TD_NOTHR | Either there is no thread with the given thread id (<code>td_ta_map_id2thr</code>)or no thread is currently executing on the given LWP (<code>td_ta_map_lwp2thr</code>). | TD_ERR | The call did not complete successfully. |
| TD_OK | The call completed successfully. | | | | | | | | | | | | |
| TD_BADTA | An invalid internal process handle was passed in. | | | | | | | | | | | | |
| TD_BADPH | There is a NULL external process handle associated with this internal process handle. | | | | | | | | | | | | |
| TD_DBERR | A call to one of the imported interface routines failed. | | | | | | | | | | | | |
| TD_NOTHR | Either there is no thread with the given thread id (<code>td_ta_map_id2thr</code>)or no thread is currently executing on the given LWP (<code>td_ta_map_lwp2thr</code>). | | | | | | | | | | | | |
| TD_ERR | The call did not complete successfully. | | | | | | | | | | | | |
| ATTRIBUTES | See <code>attributes(5)</code> for description of the following attributes: | | | | | | | | | | | | |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

`libthread_db(3T)`, `thr_create(3T)`, `thr_self(3T)`, `libthread_db(4)`, `attributes(5)`

| | |
|----------------------|--|
| NAME | td_ta_new, td_ta_delete, td_ta_get_ph – allocate and deallocate process handles for libthread_db |
| SYNOPSIS | <pre>cc [flag ...] file .../lib/libthread_db.so.1 [library ...] #include <proc_service.h> #include <thread_db.h> td_err_e td_ta_new(const struct ps_prochandle * ph_p, td_thragent_t ** ta_pp); td_err_e td_ta_delete(const td_thragent_t * ta_p); td_err_e td_ta_get_ph(const td_thragent_t * ta_p, struct ps_prochandle ** ph_pp);</pre> |
| DESCRIPTION | <p>td_ta_new() registers a target process with <code>libthread_db</code> and allocates an internal process handle of type <code>td_thragent_t</code> for this target process. Subsequent calls to <code>libthread_db</code> can use this handle to refer to this target process.</p> <p>There are actually two process handles, an internal process handle assigned by <code>libthread_db</code> and an external process handle assigned by the <code>libthread_db</code> client. There is a one-to-one correspondence between the two handles. When the client calls a <code>libthread_db</code> routine, it uses the internal process handle. When <code>libthread_db</code> calls one of the client-provided routines listed in <code>proc_service(3T)</code>, it uses the external process handle.</p> <p><code>ph</code> is the external process handle that <code>libthread_db</code> should use to identify this target process to the controlling process when it calls routines in the imported interface.</p> <p>If this call is successful, the value of the newly allocated <code>td_thragent_t</code> handle is returned in <code>* ta_pp</code>. td_ta_delete() deregisters a target process with <code>libthread_db</code>, which deallocates its internal process handle and frees any other resources <code>libthread_db</code> has acquired with respect to the target process. <code>ta_p</code> specifies the target process to be deregistered.</p> <p>td_ta_get_ph() returns in <code>* ph_pp</code> the external process handle that corresponds to the internal process handle <code>ta_p</code>. This is useful for checking internal consistency.</p> |
| RETURN VALUES | <pre>TD_OK</pre> <p>The call completed successfully.</p> |

| | |
|----------------|--|
| TD_BADPH | A NULL external process handle was passed in to <code>td_ta_new</code> . |
| TD_ERR | <code>ta_pp</code> is NULL, or an internal error occurred. |
| TD_DBERR | A call to one of the imported interface routines failed. |
| TD_MALLOC | Memory allocation failure. |
| TD_NOLIBTHREAD | The target process does not appear to be multithreaded. |

ATTRIBUTES

See `attributes(5)` for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

`libthread_db(3T)`, `proc_service(3T)`, `libthread_db(4)`, `attributes(5)`

NAME td_ta_setconcurrency – set concurrency level for target process

SYNOPSIS

```
cc [ flag... ] file.../lib/libthread_db.so.1 [ library... ]
#include <proc_service.h>
#include <thread_db.h>

td_err_e td_ta_setconcurrency(const td_thragent_t *ta_p, int level);
```

DESCRIPTION **td_ta_setconcurrency()** sets the desired concurrency level for the process identified by *ta_p* to level, just as if a thread within the process had called **thr_setconcurrency()**. See **thr_setconcurrency(3T)**.

RETURN VALUES

TD_OK The call completed successfully.

TD_BADTA An invalid internal process handle was passed in.

TD_BADPH There is a NULL external process handle associated with this internal process handle. TD_NOCAPAB The client did not implement the **ps_kill()** routine in the imported interface. See **ps_kill(3T)**.

TD_DBERR A call to one of the imported interface routines failed.

TD_ERR A libthread_db internal error occurred.

ATTRIBUTES See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO **libthread_db(3T)**, **ps_kill(3T)**, **thr_setconcurrency(3T)**, **libthread_db(4)**, **attributes(5)**

| | |
|--------------------|---|
| NAME | td_ta_sync_iter, td_ta_thr_iter, td_ta_tsd_iter – iterator functions on process handles from libthread_db library of interfaces |
| SYNOPSIS | <pre> cc [flag ...] file .../lib/libthread_db.so.1 [library ...] #include <proc_service.h> #include <thread_db.h> td_err_e td_ta_sync_iter(const td_thragent_t * ta_p, td_sync_iter_f * cb, void * cbdata_p); td_err_e td_ta_tsd_iter(const td_thragent_t * ta_p, td_key_iter_f * cb, void * cbdata_p); td_err_e td_ta_sync_iter(const td_thragent_t * ta_p, td_sync_iter_f * cb, void * cbdata_p); </pre> |
| DESCRIPTION | <p>td_ta_sync_iter() , td_ta_thr_iter() , and td_ta_tsd_iter() are iterator functions that when given a target process handle as an argument, return sets of handles for objects associated with the target process. The method is to call back a client-provided function once for each associated object, passing back a handle as well as the client-provided pointer <i>cb_data_p</i> . This enables a client to easily build a linked list of the associated objects.</p> <p>td_ta_sync_iter() returns handles of synchronization objects (mutexes, preader-writer locks, semaphores, and condition variables) associated with a process. Some synchronization objects may not be known to libthread_db() and will not be returned. If the process has initialized the synchronization object (by calling mutex_init() , for example) or a thread in the process has blocked on this object after libthread_db() attached to the synchronization object, then a handle for the synchronization object will be returned by libthread_db() . See td_sync_get_info(3T) to see operations that can be performed on synchronization object handles.</p> <p>td_ta_thr_iter() returns handles for threads that are part of the target process. For td_ta_thr_iter() , the caller specifies several criteria to select a subset of threads for which the callback function should be called. Any of these selection criteria may be wild-carded. If all of them are wild-carded, then handles for all threads in the process will be returned.</p> <p>The selection parameters and corresponding wild-card values are:</p> <pre>state (TD_THR_ANY_STATE):</pre> |

Select only threads whose state matches *state* . See **td_thr_get_info(3T)** for a list of thread states.

ti_pri (TD_THR_LOWEST_PRIORITY):

Select only threads for which the priority is at least *ti_pri* .

ti_sigmask_p (TD_SIGNO_MASK):

Select only threads whose signal mask exactly matches * *ti_sigmask_p* .

ti_user_flags (TD_THR_ANY_USER_FLAGS):

Select only threads whose user flags (specified at thread creation time) exactly match *ti_user_flags* .

td_ta_tsd_iter() returns the thread-specific data keys in use by the current process. Thread-specific data for a particular thread and key may be obtained by calling **td_thr_tsd(3T)** .

RETURN VALUES

| | |
|----------|--|
| TD_OK | The call completed successfully. |
| TD_BADTA | An invalid process handle was passed in. |
| TD_DBERR | A call to one of the imported interface routines failed. |
| TD_ERR | The call did not complete successfully. |

ATTRIBUTES

See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

libthread_db(3T) , **td_sync_get_info(3T)** , **td_thr_get_info(3T)** , **td_thr_tsd(3T)** , **libthread_db(4)** , **attributes(5)**

| | | | | | | | | | | | |
|----------------------|--|-------|----------------------------------|----------|---|----------|--|------------|--|--------|---|
| NAME | td_thr_dbsuspend, td_thr_dbresume – suspend and resume threads in libthread_db | | | | | | | | | | |
| SYNOPSIS | <pre>cc [flag ...] file .../lib/libthread_db.so.1 [library ...] #include <proc_service.h> #include <thread_db.h> td_err_e td_thr_dbsuspend(const td_thrhandle_t * th_p); td_err_e td_thr_dbresume(const td_thrhandle_t * th_p);</pre> | | | | | | | | | | |
| DESCRIPTION | <p>These operations suspend and resume the thread identified by <i>th_p</i>. A thread that has been suspended with td_thr_dbsuspend() is said to be in the "dbsuspended" state. A thread whose "dbsuspended" flag is set will not execute. If an unbound thread enters the "dbsuspended" state and is currently assigned to a lightweight process (LWP), then the LWP becomes available for assignment to a different thread.</p> <p>A thread's "dbsuspended" state is independent of the suspension state controlled by calls to thr_suspend(3T) and thr_continue(3T) from within the target process. Calling thr_continue(3T) within the target process on a thread that has been suspended during a call to td_thr_dbsuspend() will not cause that thread to resume execution; only a call to td_thr_dbresume() will do that.</p> | | | | | | | | | | |
| RETURN VALUES | <table border="0"> <tr> <td style="padding-right: 20px;">TD_OK</td> <td>The call completed successfully.</td> </tr> <tr> <td>TD_BADTH</td> <td>An invalid thread handle was passed in.</td> </tr> <tr> <td>TD_DBERR</td> <td>A call to one of the imported interface routines failed.</td> </tr> <tr> <td>TD_NOCAPAB</td> <td>The "agent thread" in the target process has not completed initialization, so this operation cannot be performed. The operation can be performed after the target process has been allowed to make some forward progress. See also libthread_db(3T)</td> </tr> <tr> <td>TD_ERR</td> <td>A libthread_db internal error occurred.</td> </tr> </table> | TD_OK | The call completed successfully. | TD_BADTH | An invalid thread handle was passed in. | TD_DBERR | A call to one of the imported interface routines failed. | TD_NOCAPAB | The "agent thread" in the target process has not completed initialization, so this operation cannot be performed. The operation can be performed after the target process has been allowed to make some forward progress. See also libthread_db(3T) | TD_ERR | A libthread_db internal error occurred. |
| TD_OK | The call completed successfully. | | | | | | | | | | |
| TD_BADTH | An invalid thread handle was passed in. | | | | | | | | | | |
| TD_DBERR | A call to one of the imported interface routines failed. | | | | | | | | | | |
| TD_NOCAPAB | The "agent thread" in the target process has not completed initialization, so this operation cannot be performed. The operation can be performed after the target process has been allowed to make some forward progress. See also libthread_db(3T) | | | | | | | | | | |
| TD_ERR | A libthread_db internal error occurred. | | | | | | | | | | |

ATTRIBUTES

See `attributes(5)` for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

`libthread_db(3T)`, `thr_continue(3T)`, `thr_suspend(3T)`,
`libthread_db(4)`, `attributes(5)`

| | |
|--------------------|---|
| NAME | td_thr_getgregs, td_thr_setgregs, td_thr_getfpregs, td_thr_setfpregs, td_thr_getxregsize, td_thr_getxregs, td_thr_setxregs – reading and writing thread registers in libthread_db |
| SYNOPSIS | <pre> cc [flag ...] file .../lib/libthread_db.so.1 [library ...] #include <proc_service.h> #include <thread_db.h> td_err_e td_thr_getgregs(const td_thrhandle_t * th_p, pgregset_t gregset); td_err_e td_thr_setgregs(const td_thrhandle_t * th_p, pgregset_t gregset); td_err_e td_thr_getfpregs(const td_thrhandle_t * th_p, prfpregset_t * fpregset); td_err_e td_thr_setfpregs(const td_thrhandle_t * th_p, prfpregset_t * fpregset); td_err_e td_thr_getxregsize(const td_thrhandle_t * th_p, int * xregsize); td_err_e td_thr_getxregs(const td_thrhandle_t * th_p, prxregset_t * xregset); td_err_e td_thr_setxregs(const td_thrhandle_t * th_p, prxregset_t * xregset); </pre> |
| DESCRIPTION | <p>These routines read and write the register sets associated with thread <i>th_p</i>. td_thr_getgregs() and td_thr_setgregs() get and set, respectively, the general registers of thread <i>th_p</i>. td_thr_getfpregs() and td_thr_setfpregs() get and set, respectively, the thread's floating point register set. td_thr_getxregsize(), td_thr_getxregs(), and td_thr_setxregs() are SPARC-specific. td_thr_getxregsize() returns in <i>xregsize</i> the size of the architecture-dependent extra state registers. td_thr_getxregs() and td_thr_setxregs() get and set, respectively, those extra state registers. On non-SPARC architectures, these routines return <code>TD_NOXREGS</code>.</p> <p>If thread <i>th_p</i> is currently executing on a lightweight process (LWP), these routines will read or write, respectively, the appropriate register set to the LWP using the imported interface. If the thread is not currently executing on a LWP, then the floating point and extra state registers may not be read or written. Some of the general registers may also not be readable or writable, depending on the architecture. In this case, td_thr_getfpregs() and td_thr_setfpregs() will return <code>TD_NOFPREGS</code>, and td_thr_getxregs() and td_thr_setxregs() will return <code>TD_NOXREGS</code>. Calls to td_thr_getgregs() and td_thr_setgregs() will succeed, but values returned for unreadable registers will be undefined, and values</p> |

specified for unwritable registers will be ignored. In this instance, a value of TD_PARTIALREGS will be returned. See the architecture-specific notes that follow regarding the registers that may be read and written for a thread not currently executing on a LWP.

SPARC On a thread not currently assigned to a LWP, only %i0-%i7, %l0-%l7, %g7, %pc, and %sp (%o6) may be read or written. %pc and %sp refer to the program counter and stack pointer that the thread will have when it resumes execution.

Intel x86 On a thread not currently assigned to a LWP, only %pc, %sp, %ebp, %edi, %edi, and %ebx may be read.

RETURN VALUES

| | |
|----------------|--|
| TD_OK | The call completed successfully. |
| TD_BADTH | An invalid thread handle was passed in. |
| TD_DBERR | A call to one of the imported interface routines failed. |
| TD_PARTIALREGS | Because the thread is not currently assigned to a LWP, not all registers were read or written. See DESCRIPTION for a discussion about which registers are not saved when a thread is not assigned to an LWP. |
| TD_NOFPREGS | Floating point registers could not be read or written, either because the thread is not currently assigned to an LWP, or because the architecture does not have such registers. |
| TD_NOXREGS | Architecture-dependent extra state registers could not be read or written, either because the thread is not currently assigned to an LWP, or because the architecture does not have such registers, or because the architecture is not a SPARC architecture. |
| TD_ERR | A libthread_db internal error occurred. |

ATTRIBUTES

See `attributes(5)` for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO | `libthread_db(3T)`, `libthread_db(4)`, `attributes(5)`

| | |
|--------------------|---|
| NAME | td_thr_get_info – get thread information in libthread_db library of interfaces |
| SYNOPSIS | cc [<i>flag...</i>] <i>file...</i> /lib/libthread_db.so.1 [<i>library...</i>] |
| DESCRIPTION | <pre>#include <proc_service.h> #include <thread_db.h> td_err_e td_thr_get_info(const td_thrhandle_t *th_p, td_thrinfo_t *ti_p);</pre> <p>The td_thr_get_info() routine fills in the <code>td_thrinfo_t</code> structure <i>ti_p</i> with values for the thread identified by <i>th_p</i>.</p> <p>The <code>td_thrinfo_t</code> structure contains the following fields:</p> <pre>typedef struct td_thrinfo_t { td_thragen_tx *ti_ta_p /* internal process handle */ unsigned ti_user_flags; /* value of flags parameter */ thread_t ti_tid; /* thread identifier */ char *ti_tls; /* pointer to thread-local storage*/ paddr ti_startfunc; /* address of function at which thread execution began*/ paddr ti_stkbase; /* base of thread's stack area*/ int ti_stksize; /* size in bytes of thread's allocated stack region*/ paddr ti_ro_area; /* address of uthread_t structure*/ int ti_ro_size /* size of the uthread_t structure in bytes */ td_thr_state_e ti_state /* state of the thread */ uchar_t ti_db_suspended /* non-zero if thread suspended by td_thr_dbsuspend*/ td_thr_type_e ti_type /* type of the thread*/ int ti_pc /* value of thread's program counter*/ int ti_sp /* value of thread's stack counter*/ short ti_flags /* set of special flags used by libthread*/ int ti_pri /* priority of thread returned by thr_getprio(3T)*/ lwpid_t ti_lid /* id of light weight process (LWP) executing this thread*/ sigset_t ti_sigmask /* thread's signal mask. See thr_sigsetmask(3T)*/ u_char ti_traceme /* non-zero if event tracing is on*/ u_char_t ti_preemptflag /* non-zero if thread preempted when last active*/ u_char_t ti_pirecflag /* non-zero if thread runs priority beside regular */ sigset_t ti_pending /* set of signals pending for this thread*/ td_thr_events_t ti_events /* bitmap of events enabled for this thread*/ } ;</pre> <p><code>td_thragent_t *ti_ta_p</code> is the internal process handle identifying the process of which the thread is a member.</p> |

unsigned `ti_user_flags` is the value of the `flags` parameter passed to `thr_create(3T)` when the thread was created.

`thread_t ti_tid` is the thread identifier for the thread returned by `libthread` when created with `thr_create(3T)`.

char *`ti_tls` is the thread's pointer to thread-local storage.

`psaddr_t ti_startfunc` is the address of the function at which thread execution began, as specified when the thread was created with `thr_create(3T)`.

`psaddr_t ti_stkbase` is the base of the thread's stack area.

int `ti_stksize` is the size in bytes of the thread's allocated stack region.

`psaddr_t ti_ro_area` is the address of the `libthread-internal uthread_t` structure for this thread. Since accessing the `uthread_t` structure directly violates the encapsulation provided by `libthread_db`, this field should generally not be used. However, it may be useful as a prototype for extensions.

`td_thr_state_e ti_state` is the state in which the thread is. The `td_thr_state_e` enumeration type may contain the following values:

| | |
|-------------------------------|---|
| <code>TD_THR_ANY_STATE</code> | Never returned by <code>td_thr_get_info</code> . <code>TD_THR_ANY_STATE</code> is used as a wildcard to select threads in <code>td_ta_thr_iter()</code> . |
| <code>TD_THR_UNKNOWN</code> | <code>libthread_db</code> cannot determine the state of the thread. |
| <code>TD_THR_STOPPED</code> | The thread has been stopped by a call to <code>thr_suspend(3T)</code> . |
| <code>TD_THR_RUN</code> | The thread is runnable, but it is not currently assigned to a LWP. |
| <code>TD_THR_ACTIVE</code> | The thread is currently executing on a LWP. |
| <code>TD_THR_ZOMBIE</code> | The thread has exited, but it has not yet been deallocated by a call to <code>thr_join(3T)</code> . |
| <code>TD_THR_SLEEP</code> | The thread is not currently runnable. |

`TD_THR_STOPPED_ASLEEP` The thread is both blocked by `TD_THR_SLEEP`, and stopped by a call to `td_thr_dbsuspend(3T)`.

`uchar_t ti_db_suspended` is non-zero if and only if this thread is currently suspended because the controlling process has called `td_thr_dbsuspend` on it.

`td_thr_type_e ti_type` is a type of thread. It will be either `TD_THR_USER` for a user thread (one created by the application), or `TD_THR_SYSTEM` for one created by `libthread`.

`int ti_pc` is the value of the thread's program counter, provided that the thread's `ti_state` value is `TD_THR_SLEEP`, `TD_THR_STOPPED`, or `TD_THR_STOPPED_ASLEEP`. Otherwise, the value of this field is undefined.

`int ti_sp` is the value of the thread's stack pointer, provided that the thread's `ti_state` value is `TD_THR_SLEEP`, `TD_THR_STOPPED`, or `TD_THR_STOPPED_ASLEEP`. Otherwise, the value of this field is undefined.

`short ti_flags` is a set of special flags used by `libthread`, currently of use only to those debugging `libthread`.

`int ti_pri` is the thread's priority, as it would be returned by `thr_getprio(3T)`.

`lwpid_t ti_lid` is the ID of the LWP executing this thread, or the ID of the LWP that last executed this thread, if this thread is not currently assigned to a LWP.

`sigset_t ti_sigmask` is this thread's signal mask. See `thr_sigsetmask(3T)`.

`u_char ti_traceme` is non-zero if and only if event tracing for this thread is on.

`uchar_t ti_preemptflag` is non-zero if and only if the thread was preempted the last time it was active.

`uchar_t ti_pirecflag` is non-zero if and only if due to priority inheritance the thread is currently running at a priority other than its regular priority.

`td_thr_events_t ti_events` is the bitmap of events enabled for this thread.

RETURN VALUES

`TD_OK` The call completed successfully.

`TD_BADTH` An invalid thread handle was passed in.

TD_DBERR A call to one of the imported interface routines failed.

TD_ERR The call did not complete successfully.

ATTRIBUTES

See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

libthread(3T), **libthread_db(3T)**, **td_ta_thr_iter(3T)**,
td_thr_dbsuspend(3T), **thr_create(3T)**, **thr_getprio(3T)**,
thr_join(3T), **thr_sigsetmask(3T)**, **thr_suspend(3T)**, **libthread(4)**,
libthread_db(4), **attributes(5)**

NAME td_thr_lockowner – iterate over the set of locks owned by a thread

SYNOPSIS cc [*flag...*] *file...*/lib/libthread_db.so.1 [*library...*]

```
#include <proc_service.h>
#include <thread_db.h>

td_err_e td_thr_lockowner(const td_thrhandle_t *th_p, td_sync_iter_f *cb, void
*cb_data_p);
```

DESCRIPTION **td_thr_lockowner()** calls the iterator function *cb* once for every mutex that is held by the thread whose handle is *th_p*. The synchronization handle and the pointer *cb_data_p* are passed to the function. See **td_ta_thr_iter(3T)** for a similarly structured function.

Iteration terminates early if the callback function *cb* returns a non-zero value.

RETURN VALUES

| | |
|----------|---|
| TD_OK | The call completed successfully. |
| TD_BADTH | An invalid thread handle was passed in. |
| TD_BADPH | There is a NULL external process handle associated with this internal process handle. |
| TD_DBERR | A call to one of the imported interface routines failed. |
| TD_ERR | A libthread_db internal error occurred. |

ATTRIBUTES See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO **libthread_db(3T)**, **td_ta_thr_iter(3T)**, **libthread_db(4)**, **attributes(5)**

NAME td_thr_setprio – set the priority of a thread

SYNOPSIS

```
cc [ flag... ] file.../lib/libthread_db.so.1 [ library... ]
#include <proc_service.h>
#include <thread_db.h>
td_err_e td_thr_setprio(const td_thrhandle_t *th_p, const int new_prio);
```

DESCRIPTION **td_thr_setprio()** sets thread *th_p*'s priority to *new_prio*, just as if a thread within the process had called `thr_setprio()`. See **thr_setprio(3T)**.

RETURN VALUES

TD_OK The call completed successfully.

TD_BADTH An invalid thread handle was passed in.

TD_DBERR A call to one of the imported interface routines failed.

TD_ERR *new_prio* is an illegal value (out of range).

ATTRIBUTES See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO **libthread_db(3T)**, **thr_setprio(3T)**, **libthread_db(4)**, **attributes(5)**

| | |
|----------------------|---|
| NAME | td_thr_setsigpending, td_thr_sigsetmask – manage thread signals for libthread_db |
| SYNOPSIS | <pre>cc [flag ...] file .../lib/libthread_db.so.1 [library ...] #include <proc_service.h> #include <thread_db.h> td_err_e td_thr_setsigpending(const td_thrhandle_t * th_p, const uchar_t ti_sigpending_flag, const sigset_t ti_sigmask); td_err_e td_thr_sigsetmask(const td_thrhandle_t * th_p, const sigset_t ti_sigmask);</pre> |
| DESCRIPTION | <p>The td_thr_setsigpending() and td_thr_sigsetmask() operations affect the signal state of the thread identified by <i>th_p</i> .</p> <p>td_thr_setsigpending() sets the set of pending signals for thread <i>th_p</i> to <i>ti_sigpending</i> . The value of the <code>libthread</code> -internal field that indicates whether a thread has any signal pending is set to <i>ti_sigpending_flag</i> . To be consistent, <i>ti_sigpending_flag</i> should be zero if and only if all of the bits in <i>ti_sigpending</i> are zero.</p> <p>td_thr_sigsetmask() sets the signal mask of the thread <i>th_p</i> as if the thread had set its own signal mask by way of thr_sigsetmask(3T) . The new signal mask is the value of <i>ti_sigmask</i> .</p> <p>There is no equivalent to the <code>SIG_BLOCK</code> or <code>SIG_UNBLOCK</code> operations of thr_sigsetmask(3T) , which mask or unmask specific signals without affecting the mask state of other signals. To block or unblock specific signals, either stop the whole process, or the thread, if necessary, by td_thr_dbsuspend() . Then determine the thread's existing signal mask by calling td_thr_get_info() and reading the <i>ti_sigmask</i> field of the <code>td_thrinfo_t</code> structure returned. Modify it as desired, and set the new signal mask with td_thr_sigsetmask() .</p> |
| RETURN VALUES | <pre>TD_OK The call completed successfully. TD_BADTH An invalid thread handle was passed in. TD_DBERR A call to one of the imported interface routines failed.</pre> |

TD_ERR A libthread_db internal error occurred.

ATTRIBUTES

See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

libthread_db(3T), **td_thr_dbsuspend(3T)**, **td_thr_get_info(3T)**,
libthread_db(4), **attributes(5)**

NAME | td_thr_sleepinfo – return the synchronization handle for the object on which a thread is blocked

SYNOPSIS | cc [*flag...*] *file...*/lib/libthread_db.so.1 [*library...*]

```
#include <proc_service.h>
#include <thread_db.h>

td_err_e td_thr_sleepinfo(const td_thrhandle_t *th_p, td_synchronize_t *sh_p);
```

DESCRIPTION | **td_thr_sleepinfo()** returns in **sh_p* the handle of the synchronization object on which a sleeping thread is blocked.

RETURN VALUES

| | |
|----------|---|
| TD_OK | The call completed successfully. |
| TD_BADTH | An invalid thread handle was passed in. |
| TD_DBERR | A call to one of the imported interface routines failed. |
| TD_ERR | The thread <i>th_p</i> is not blocked on a synchronization object, or a libthread_db internal error occurred. |

ATTRIBUTES | See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO | **libthread_db(3T)**, **libthread_db(4)**, **attributes(5)**

NAME td_thr_tsd – get a thread’s thread-specific data for libthread_db library of interfaces

SYNOPSIS cc [*flag...*] *file...*/lib/libthread_db.so.1 [*library...*]

```
#include <proc_service.h>
#include <thread_db.h>

td_err_e td_thr_tsd(const td_thrhandle_t, const thread_key_t key, void *data_pp);
```

DESCRIPTION **td_thr_tsd()** returns in **data_pp* the thread-specific data pointer for the thread identified by *th_p* and the thread-specific data key *key*. This is the same value that thread *th_p* would obtain if it called **thr_getspecific(3T)**.

To find all the thread-specific data keys in use in a given target process, call **td_ta_tsd_iter(3T)**.

RETURN VALUES

| | |
|----------|--|
| TD_OK | The call completed successfully. |
| TD_BADTH | An invalid thread handle was passed in. |
| TD_DBERR | A call to one of the imported interface routines failed. |
| TD_ERR | A libthread_db internal error occurred. |

ATTRIBUTES See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO **libthread_db(3T)**, **td_ta_tsd_iter(3T)**, **thr_getspecific(3T)**, **libthread_db(4)**, **attributes(5)**

NAME td_thr_validate – test a thread handle for validity

SYNOPSIS

```
cc [ flag... ] file.../lib/libthread_db.so.1 [ library... ]
#include <proc_service.h>
#include <thread_db.h>

td_err_e td_thr_validate(const td_thrhandle_t *th_p);
```

DESCRIPTION **td_thr_validate()** tests whether *th_p* is a valid thread handle. A valid thread handle may become invalid if its thread exits.

RETURN VALUES

TD_OK The call completed successfully. *th_p* is a valid thread handle.

TD_BADTH *th_p* was NULL.

TD_DBERR A call to one of the imported interface routines failed.

TD_NOTHR *th_p* is not a valid thread handle.

TD_ERR A libthread_db internal error occurred.

ATTRIBUTES See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO **libthread_db(3T)**, **libthread_db(4)**, **attributes(5)**

NAME tell – return a file offset for a file descriptor

SYNOPSIS #include <unistd.h>
off_t tell(int *fd*);

DESCRIPTION The **tell()** function obtains the current value of the file-position indicator for the file descriptor *fd*.

RETURN VALUES Upon successful completion, **tell()** returns the current value of the file-position indicator for *fd* measured in bytes from the beginning of the file.
Otherwise, it returns `-1` and sets `errno` to indicate the error.

ERRORS The **tell()** function will fail if:

EBADF The file descriptor *fd* is not an open file descriptor.

EOVERFLOW The current file offset cannot be represented correctly in an object of type `off_t`.

ESPIPE The file descriptor *fd* is associated with a pipe or FIFO.

USAGE The **tell()** function is equivalent to `lseek(fd, 0, SEEK_CUR)`.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO `lseek(2)`, **attributes(5)**

NAME | telldir – current location of a named directory stream

SYNOPSIS | #include <dirent.h>
 | long int **telldir**(DIR **dirp*);

DESCRIPTION | The **telldir()** function obtains the current location associated with the directory stream specified by *dirp*.
 | If the most recent operation on the directory stream was a **seekdir(3C)**, the directory position returned from the **telldir()** is the same as that supplied as a *loc* argument for **seekdir()**.

RETURN VALUES | Upon successful completion, **telldir()** returns the current location of the specified directory stream.

ERRORS | No errors are defined.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO | **opendir(3C)**, **readdir(3C)**, **seekdir(3C)**, **attributes(5)**

| | |
|----------------------|---|
| NAME | termattrs – return the video attributes supported by the terminal |
| SYNOPSIS | <pre>#include <curses.h> attr_t termattrs(void);</pre> |
| DESCRIPTION | The termattrs() function determines which video attributes are supported by the terminal. |
| RETURN VALUES | The termattrs() function returns a logical OR of all video attributes available on the terminal. |
| ERRORS | None. |
| SEE ALSO | attr_get(3XC) , attroff(3XC) |

| | |
|--------------------|--|
| NAME | termios – general terminal interface |
| SYNOPSIS | <pre>#include <termios.h> int tcgetattr(int <i>fildev</i>, struct termios *<i>termios_p</i>); int tcsetattr(int <i>fildev</i>, int <i>optional_actions</i>, const struct termios *<i>termios_p</i>); int tcsendbreak(int <i>fildev</i>, int <i>duration</i>); int tcdrain(int <i>fildev</i>); int tcflush(int <i>fildev</i>, int <i>queue_selector</i>); int tcflow(int <i>fildev</i>, int <i>action</i>); speed_t cfgetospeed(const struct termios *<i>termios_p</i>); int cfsetospeed(struct termios *<i>termios_p</i>, speed_t <i>speed</i>); speed_t cfgetispeed(const struct termios *<i>termios_p</i>); int cfsetispeed(struct termios *<i>termios_p</i>, speed_t <i>speed</i>); #include <sys/types.h> #include <termios.h> pid_t tcgetpgrp(int <i>fildev</i>); int tcsetpgrp(int <i>fildev</i>, pid_t <i>pgid</i>); pid_t tcgetsid(int <i>fildev</i>);</pre> |
| DESCRIPTION | <p>These functions describe a general terminal interface for controlling asynchronous communications ports. A more detailed overview of the terminal interface can be found in <code>termio(7I)</code>, which also describes an <code>ioctl(2)</code> interface that provides the same functionality. However, the function interface described by these functions is the preferred user interface.</p> <p>Each of these functions is now described on a separate manual page.</p> |
| SEE ALSO | <p><code>ioctl(2)</code>, <code>cfgetispeed(3)</code>, <code>cfgetospeed(3)</code>, <code>cfsetispeed(3)</code>, <code>cfsetospeed(3)</code>, <code>tcdrain(3)</code>, <code>tcflow(3)</code>, <code>tcflush(3)</code>, <code>tcgetattr(3)</code>, <code>tcgetpgrp(3)</code>, <code>tcgetsid(3)</code>, <code>tcsendbreak(3)</code>, <code>tcsetattr(3)</code>, <code>tcsetpgrp(3)</code>, <code>tcsendbreak(3)</code>, <code>termio(7I)</code></p> |

| | |
|----------------------|--|
| NAME | termname - return the value of the environmental variable TERM |
| SYNOPSIS | <pre>#include <curses.h> char *termname(void);</pre> |
| DESCRIPTION | The termname() function returns a pointer to the value of the environmental variable TERM (truncated to 14 characters). |
| RETURN VALUES | The termname() returns a pointer to the terminal's name. |
| ERRORS | None. |
| SEE ALSO | del_curterm(3XC) |

| | |
|---------------------------------|---|
| NAME | t_errno – XTI error return value |
| SYNOPSIS | #include <xti.h> |
| DESCRIPTION | <p>This error return value is part of the XTI interfaces that evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI interface that has the same name as an XTI interfaces, a different headerfile, <tiuser.h>, must be used. Refer the the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>t_errno is used by XTI functions to return error values.</p> <p>XTI functions provide an error number in t_errno which has type <i>int</i> and is defined in <xti.h>. The value of t_errno will be defined only after a call to a XTI function for which it is explicitly stated to be set and until it is changed by the next XTI function call. The value of t_errno should only be examined when it is indicated to be valid by a function's return value. Programs should obtain the definition of t_errno by the inclusion of <xti.h>. The practice of defining t_errno in program as <code>extern int t_errno</code> is obsolescent. No XTI function sets t_errno to 0 to indicate an error.</p> <p>It is unspecified whether t_errno is a macro or an identifier with external linkage. It represents a modifiable lvalue of type <i>int</i>. If a macro definition is suppressed in order to access an actual object or a program defines an identifier with name <i>t_errno</i>, the behavior is undefined.</p> <p>The symbolic values stored in t_errno by an XTI function are defined in the ERRORS sections in all relevant XTI function definition pages.</p> |
| TLI COMPATIBILITY | <p>t_errno is also used by TLI functions to return error values.</p> <p>The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.</p> |
| Interface Header | <p>The XTI interfaces use the header file, <xti.h>. TLI interfaces should <i>not</i> use this header. They should use the header:</p> <pre>#include <tiuser.h></pre> |
| Error Description Values | <p>The t_errno values that can be set by the XTI interface but cannot be set by the TLI interface are:</p> <pre>TNOSTRUCTYPE</pre> |

TBADNAME
TBADQLEN
TADDRBUSY
TINDOUT
TPROVMISMATCH
TRESADDR
TQFULL
TPROTO

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

attributes(5)

Transport Interfaces Programming Guide

| | |
|--------------------|---|
| NAME | t_error – produce error message |
| SYNOPSIS | <pre>#include <xti.h></pre> <pre>int t_error(const char *errmsg);</pre> |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>The <code>t_error()</code> function produces a message on the standard error output which describes the last error encountered during a call to a transport function. The argument string <code>errmsg</code> is a user-supplied error message that gives context to the error.</p> <p>The error message is written as follows: first (if <code>errmsg</code> is not a null pointer and the character pointed to by <code>errmsg</code> is not the null character) the string pointed to by <code>errmsg</code> followed by a colon and a space; then a standard error message string for the current error defined in <code>t_errno</code>. If <code>t_errno</code> has a value different from <code>TSYSERR</code>, the standard error message string is followed by a newline character. If, however, <code>t_errno</code> is equal to <code>TSYSERR</code>, the <code>t_errno</code> string is followed by the standard error message string for the current error defined in <code>errno</code> followed by a newline.</p> <p>The language for error message strings written by <code>t_error()</code> is that of the current locale. If it is English, the error message string describing the value in <code>t_errno</code> may be derived from the comments following the <code>t_errno</code> codes defined in <code>xti.h</code>. The contents of the error message strings describing the value in <code>errno</code> are the same as those returned by the <code>strerror(3C)</code> function with an argument of <code>errno</code>.</p> <p>The error number, <code>t_errno</code>, is only set when an error occurs and it is not cleared on successful calls.</p> |
| EXAMPLES | <p>If a <code>t_connect(3N)</code> function fails on transport endpoint <code>fd2</code> because a bad address was given, the following call might follow the failure:</p> <pre>t_error("t_connect failed on fd2");</pre> <p>The diagnostic message to be printed would look like:</p> |

```
t_connect failed on fd2: incorrect addr format
```

where *incorrect addr format* identifies the specific error that occurred, and *t_connect failed on fd2* tells the user which function failed on which transport endpoint.

RETURN VALUES

Upon completion, a value of 0 is returned.

VALID STATES

All - apart from T_UNINIT

ERRORS

No errors are defined for the **t_error()** function.

**TLI
COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```
#include <tiuser.h>
```

**Error Description
Values**

The `t_errno` value that can be set by the XTI interface and cannot be set by the TLI interface is:

```
TPROTO
```

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

t_errno(3N) **strerror(3C)**, **attributes(5)**

Transport Interfaces Programming Guide

| | |
|----------------------|---|
| NAME | t_free – free a library structure |
| SYNOPSIS | #include <xti.h> |
| DESCRIPTION | <p>int t_free(void *ptr, int struct_type);</p> <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>The t_free() function frees memory previously allocated by t_alloc(3N). This function will free memory for the specified structure, and will also free memory for buffers referenced by the structure.</p> <p>The argument <i>ptr</i> points to one of the seven structure types described for t_alloc(3N), and <i>struct_type</i> identifies the type of that structure which must be one of the following:</p> <pre> T_BIND struct t_bind T_CALL struct t_call T_OPTMGMT struct t_optmgmt T_DIS struct t_discon T_UNITDATA struct t_unitdata T_UDERROR struct t_uderr T_INFO struct t_info </pre> <p>where each of these structures is used as an argument to one or more transport functions.</p> <p>The function t_free() will check the <i>addr</i>, <i>opt</i> and <i>udata</i> fields of the given structure, as appropriate, and free the buffers pointed to by the <i>buf</i> field of the <code>netbuf</code> structure. If <i>buf</i> is a null pointer, t_free() will not attempt to free memory. After all buffers are freed, t_free() will free the memory associated with the structure pointed to by <i>ptr</i>.</p> <p>Undefined results will occur if <i>ptr</i> or any of the <i>buf</i> pointers points to a block of memory that was not previously allocated by t_alloc(3N).</p> |
| RETURN VALUES | Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and <code>t_errno</code> is set to indicate an error. |

VALID STATES

ALL - apart from T_UNINIT.

ERRORS

On failure, `t_errno` is set to the following:

TNOSTRUCTYPE

Unsupported *struct_type* requested.

TPROTO

This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (*t_errno*).

TSYSERR

A system error has occurred during execution of this function.

TLI COMPATIBILITY

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```
#include <tiuser.h>
```

Error Description Values

The `t_errno` value that can be set by the XTI interface and cannot be set by the TLI interface is:

TPROTO

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`t_alloc(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

| | |
|--------------------|---|
| NAME | tgetent, tgetflag, tgetnum, tgetstr, tgoto – emulate the termcap database |
| SYNOPSIS | <pre>#include <term.h> int tgetent(char * bp, const char * name); int tgetflag(char id [2]); int tgetnum(char id [2]); char * tgetstr(char cap [2], char ** area); char * tgoto(const char * cap, int col, int row);</pre> |
| PARAMETERS | <p>bp Is a pointer to a buffer. This parameter is ignored.</p> <p>name Is the termcap entry to look up.</p> <p>cap Is the pointer to a termcap capability.</p> <p>area Is a pointer to the area where tgetstr() stores the decoded string.</p> <p>col Is the column placement of the new cursor.</p> <p>row Is the row placement of the new cursor.</p> |
| DESCRIPTION | <p>These functions provide an interface to the termcap database.</p> <p>The tgetent() function looks up the termcap entry for the terminal name. The bp parameter is ignored by this function.</p> <p>The tgetflag() function returns the Boolean value of the termcap capability pointed to by cap .</p> <p>The tgetnum() function looks up the numeric entry for cap .</p> <p>The tgetstr() function looks up the string entry for the termcap capability pointed to by cap , placing the decoded string at area and advancing the area pointers. The tputs(3XC) function should be used to output the string.</p> <p>The tgoto() function decodes cursor values returned from tgetstr() . A pointer to a cursor addressing string is returned that, when sent to the terminal with tputs() , moves the cursor to the new location.</p> <p>These functions are included for compatibility purposes with programs that use the termcap library. New programs should use terminfo functions described on the tigetflag(3XC) man page.</p> |

RETURN VALUES

On success, those functions that return integers return `OK` . Otherwise, they return `ERR` .

Those functions that return pointers return a null pointer when an error occurs.

ERRORS

None.

SEE ALSO

`putp(3XC)` , `setupterm(3XC)` , `tigetflag(3XC)`

| | |
|--------------------|---|
| NAME | t_getinfo – get protocol-specific service information |
| SYNOPSIS | <pre>#include <xti.h> int t_getinfo(int fd, struct t_info *info);</pre> |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>This function returns the current characteristics of the underlying transport protocol and/or transport connection associated with file descriptor <code>fd</code>. The <code>info</code> pointer is used to return the same information returned by <code>t_open(3N)</code>, although not necessarily precisely the same values. This function enables a transport user to access this information during any phase of communication.</p> <p>This argument points to a <code>t_info</code> structure which contains the following members:</p> <pre>t_scalar_t addr; /*max size in octets of the transport protocol address*/ t_scalar_t options; /*max number of bytes of protocol-specific options */ t_scalar_t tsdu; /*max size in octets of a transport service data unit */ t_scalar_t etsdu; /*max size in octets of an expedited transport service*/ /*data unit (ETSU) */ t_scalar_t connect; /*max number of octets allowed on connection */ /*establishment functions */ t_scalar_t discon; /*max number of octets of data allowed on t_snddis() */ /*and t_rcvdis() functions */ t_scalar_t servtype; /*service type supported by the transport provider */ t_scalar_t flags; /*other info about the transport provider */</pre> <p>The values of the fields have the following meanings:</p> <p>addr A value greater than zero indicates the maximum size of a transport protocol address and a value of <code>T_INVALID (-2)</code> specifies that the transport provider does not provide user access to transport protocol addresses.</p> <p>options A value greater than zero indicates the maximum number of bytes of protocol-specific options supported by the provider, and a value of <code>T_INVALID (-2)</code> specifies that the transport provider does not support user-settable options.</p> <p>tsdu A value greater than zero specifies the maximum size in octets of a transport service data unit (TSU); a value of <code>T_NULL (zero)</code> specifies that the transport provider does not</p> |

| | |
|-----------------|---|
| | support the concept of TSDU, although it does support the sending of a datastream with no logical boundaries preserved across a connection; a value of T_INFINITE (-1) specifies that there is no limit on the size in octets of a TSDU; and a value of T_INVALID (-2) specifies that the transfer of normal data is not supported by the transport provider. |
| etsdu | A value greater than zero specifies the maximum size in octets of an expedited transport service data unit (ETSDU); a value of T_NULL (zero) specifies that the transport provider does not support the concept of ETSDU, although it does support the sending of an expedited data stream with no logical boundaries preserved across a connection; a value of T_INFINITE (-1) specifies that there is no limit on the size (in octets) of an ETSDU; and a value of T_INVALID (-2) specifies that the transfer of expedited data is not supported by the transport provider. Note that the semantics of expedited data may be quite different for different transport providers. |
| connect | A value greater than zero specifies the maximum number of octets that may be associated with connection establishment functions and a value of T_INVALID (-2) specifies that the transport provider does not allow data to be sent with connection establishment functions. |
| discon | If the T_ORDRELDATA bit in flags is clear, a value greater than zero specifies the maximum number of octets that may be associated with the <code>t_snddis(3N)</code> and <code>t_rcvdis(3N)</code> functions, and a value of T_INVALID (-2) specifies that the transport provider does not allow data to be sent with the abortive release functions. If the T_ORDRELDATA bit is set in flags, a value greater than zero specifies the maximum number of octets that may be associated with the <code>t_sndreldata()</code> , <code>t_rcvreldata()</code> , <code>t_snddis(3N)</code> and <code>t_rcvdis(3N)</code> functions. |
| servtype | This field specifies the service type supported by the transport provider, as described below. |
| flags | This is a bit field used to specify other information about the communications provider. If the T_ORDRELDATA bit is set, the communications provider supports sending user data with an orderly release. If the T_SENZERO bit is set in flags, |

this indicates that the underlying transport provider supports the sending of zero-length TSDUs.

If a transport user is concerned with protocol independence, the above sizes may be accessed to determine how large the buffers must be to hold each piece of information. Alternatively, the `t_alloc(3N)` function may be used to allocate these buffers. An error will result if a transport user exceeds the allowed data size on any function. The value of each field may change as a result of protocol option negotiation during connection establishment (the `t_optmgmt(3N)` call has no effect on the values returned by `t_getinfo()`). These values will only change from the values presented to `t_open(3N)` after the endpoint enters the `T_DATAXFER` state.

The `servtype` field of `info` specifies one of the following values on return:

| | |
|-------------------------|---|
| <code>T_COTS</code> | The transport provider supports a connection-mode service but does not support the optional orderly release facility. |
| <code>T_COTS_ORD</code> | The transport provider supports a connection-mode service with the optional orderly release facility. |
| <code>T_CLTS</code> | The transport provider supports a connectionless-mode service. For this service type, <code>t_open(3N)</code> will return <code>T_INVALID</code> (-1) for <i>etsdu</i> , <i>connect</i> and <i>discon</i> . |

RETURN VALUES

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and `t_errno` is set to indicate an error.

VALID STATES

ALL - apart from `T_UNINIT`.

ERRORS

On failure, `t_errno` is set to one of the following:

| | |
|----------------------|---|
| <code>TBADF</code> | The specified file descriptor does not refer to a transport endpoint. |
| <code>TPROTO</code> | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). |
| <code>TSYSERR</code> | A system error has occurred during execution of this function. |

TLI COMPATIBILITY

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```
#include <tiuser.h>
```

Error Description Values

The `t_errno` value `TPROTO` can be set by the XTI interface but not by the TLI interface.

The t_info Structure

For TLI, the `t_info` structure referenced by *info* lacks the following structure member:

```
t_scalar_t flags; /* other info about the transport provider */
```

This member was added to `struct t_info` in the XTI interfaces.

When a value of `-1` is observed as the return value in various `t_info` structure members, it signifies that the transport provider can handle an infinite length buffer for a corresponding attribute, such as address data, option data, TSDU (octet size), ETSDU (octet size), connection data, and disconnection data. The corresponding structure members are `addr`, `options`, `tsdu`, `estdu`, `connect`, and `discon`, respectively.

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`t_alloc(3N)`, `t_open(3N)`, `t_optmgmt(3N)`, `t_rcvdis(3N)`, `t_snddis(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

| | | | | | | | | | |
|----------------------|---|-------|---|-----------|--|--------|---|---------|--|
| NAME | t_getprotaddr – get the protocol addresses | | | | | | | | |
| SYNOPSIS | <pre>#include <xti.h> int t_getprotaddr(int fd, struct t_bind *boundaddr, struct t_bind *peeraddr);</pre> | | | | | | | | |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>The <code>t_getprotaddr()</code> function returns local and remote protocol addresses currently associated with the transport endpoint specified by <code>fd</code>. In <code>boundaddr</code> and <code>peeraddr</code> the user specifies <code>maxlen</code>, which is the maximum size (in bytes) of the address buffer, and <code>buf</code> which points to the buffer where the address is to be placed. On return, the <code>buf</code> field of <code>boundaddr</code> points to the address, if any, currently bound to <code>fd</code>, and the <code>len</code> field specifies the length of the address. If the transport endpoint is in the <code>T_UNBND</code> state, zero is returned in the <code>len</code> field of <code>boundaddr</code>. The <code>buf</code> field of <code>peeraddr</code> points to the address, if any, currently connected to <code>fd</code>, and the <code>len</code> field specifies the length of the address. If the transport endpoint is not in the <code>T_DATAXFER</code>, <code>T_INREL</code>, <code>T_OUTCON</code> or <code>T_OUTREL</code> states, zero is returned in the <code>len</code> field of <code>peeraddr</code>. If the <code>maxlen</code> field of <code>boundaddr</code> or <code>peeraddr</code> is set to zero, no address is returned.</p> | | | | | | | | |
| RETURN VALUES | Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and <code>t_errno</code> is set to indicate the error. | | | | | | | | |
| VALID STATES | ALL - apart from <code>T_UNINIT</code> . | | | | | | | | |
| ERRORS | <p>On failure, <code>t_errno</code> is set to one of the following:</p> <table border="0"> <tr> <td style="vertical-align: top;">TBADF</td> <td>The specified file descriptor does not refer to a transport endpoint.</td> </tr> <tr> <td style="vertical-align: top;">TBUFOVFLW</td> <td>The number of bytes allocated for an incoming argument (<code>maxlen</code>) is greater than 0 but not sufficient to store the value of that argument.</td> </tr> <tr> <td style="vertical-align: top;">TPROTO</td> <td>This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>).</td> </tr> <tr> <td style="vertical-align: top;">TSYSERR</td> <td>A system error has occurred during execution of this function.</td> </tr> </table> | TBADF | The specified file descriptor does not refer to a transport endpoint. | TBUFOVFLW | The number of bytes allocated for an incoming argument (<code>maxlen</code>) is greater than 0 but not sufficient to store the value of that argument. | TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). | TSYSERR | A system error has occurred during execution of this function. |
| TBADF | The specified file descriptor does not refer to a transport endpoint. | | | | | | | | |
| TBUFOVFLW | The number of bytes allocated for an incoming argument (<code>maxlen</code>) is greater than 0 but not sufficient to store the value of that argument. | | | | | | | | |
| TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). | | | | | | | | |
| TSYSERR | A system error has occurred during execution of this function. | | | | | | | | |

**TLI
COMPATIBILITY****ATTRIBUTES**

In the TLI interface definition, no counterpart of this routine was defined.

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

t_bind(3N), **attributes(5)**

Transport Interfaces Programming Guide

| | | | | | | | | | | | | | | | |
|----------------------|---|---------|---|--------|---|------------|--|---------|------------------------------|------------|----------------|----------|--|---------|--|
| NAME | t_getstate – get the current state | | | | | | | | | | | | | | |
| SYNOPSIS | <pre>#include <xti.h> int t_getstate(int fd);</pre> | | | | | | | | | | | | | | |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>The t_getstate() function returns the current state of the provider associated with the transport endpoint specified by <i>fd</i>.</p> | | | | | | | | | | | | | | |
| RETURN VALUES | <p>State is returned upon successful completion. Otherwise, a value of -1 is returned and <code>t_errno</code> is set to indicate an error. The current state is one of the following:</p> <table border="0"> <tr> <td>T_UNBND</td> <td>Unbound.</td> </tr> <tr> <td>T_IDLE</td> <td>Idle.</td> </tr> <tr> <td>T_OUTCON</td> <td>Outgoing connection pending.</td> </tr> <tr> <td>T_INCON</td> <td>Incoming connection pending.</td> </tr> <tr> <td>T_DATAXFER</td> <td>Data transfer.</td> </tr> <tr> <td>T_OUTREL</td> <td>Outgoing direction orderly release sent.</td> </tr> <tr> <td>T_INREL</td> <td>Incoming direction orderly release received.</td> </tr> </table> <p>If the provider is undergoing a state transition when t_getstate() is called, the function will fail.</p> | T_UNBND | Unbound. | T_IDLE | Idle. | T_OUTCON | Outgoing connection pending. | T_INCON | Incoming connection pending. | T_DATAXFER | Data transfer. | T_OUTREL | Outgoing direction orderly release sent. | T_INREL | Incoming direction orderly release received. |
| T_UNBND | Unbound. | | | | | | | | | | | | | | |
| T_IDLE | Idle. | | | | | | | | | | | | | | |
| T_OUTCON | Outgoing connection pending. | | | | | | | | | | | | | | |
| T_INCON | Incoming connection pending. | | | | | | | | | | | | | | |
| T_DATAXFER | Data transfer. | | | | | | | | | | | | | | |
| T_OUTREL | Outgoing direction orderly release sent. | | | | | | | | | | | | | | |
| T_INREL | Incoming direction orderly release received. | | | | | | | | | | | | | | |
| ERRORS | <p>On failure, <code>t_errno</code> is set to one of the following:</p> <table border="0"> <tr> <td>TBADF</td> <td>The specified file descriptor does not refer to a transport endpoint.</td> </tr> <tr> <td>TPROTO</td> <td>This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>).</td> </tr> <tr> <td>TSTATECHNG</td> <td>The transport provider is undergoing a transient state change.</td> </tr> </table> | TBADF | The specified file descriptor does not refer to a transport endpoint. | TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). | TSTATECHNG | The transport provider is undergoing a transient state change. | | | | | | | | |
| TBADF | The specified file descriptor does not refer to a transport endpoint. | | | | | | | | | | | | | | |
| TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). | | | | | | | | | | | | | | |
| TSTATECHNG | The transport provider is undergoing a transient state change. | | | | | | | | | | | | | | |

TSYSERR A system error has occurred during execution of this function.

**TLI
COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```
#include <tiuser.h>
```

**Error Description
Values**

The `t_errno` value that can be set by the XTI interface and cannot be set by the TLI interface is:

TPROTO

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`t_open(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

| | |
|--------------------|--|
| NAME | thr_create – create a thread |
| SYNOPSIS | <pre>cc - mt [<i>flag...</i>] <i>file...</i>[<i>library...</i>] #include <thread.h> int thr_create(void *<i>stack_base</i>, size_t <i>stack_size</i>, void *(*<i>start_func</i>, void *), void *<i>arg</i>, long <i>flags</i>, thread_t *<i>new_thread_ID</i>);</pre> |
| DESCRIPTION | <p>Thread creation adds a new thread of control to the current process. The procedure main(), itself, is a single thread of control. Each thread executes simultaneously with all the other threads within the calling process, and with other threads from other active processes.</p> <p>A newly created thread shares all of the calling process' global data with the other threads in this process; however, it has its own set of attributes and private execution stack. The new thread inherits the calling thread's signal mask, possibly, and scheduling priority. Pending signals for a new thread are not inherited and will be empty.</p> <p>The call to create a thread takes the address of a user-defined function, specified by <i>start_func</i>, as one of its arguments, which is the complete execution routine for the new thread.</p> <p>The lifetime of a thread begins with the successful return from thr_create(), which calls <i>start_func()</i> and ends with either:</p> <ul style="list-style-type: none"> ■ the normal completion of <i>start_func()</i>, ■ the return from an explicit call to thr_exit(3T), or ■ the conclusion of the calling process (see exit(2)). <p>The new thread performs by calling the function defined by <i>start_func</i> with one argument, <i>arg</i>. If more than one argument needs to be passed to <i>start_func</i>, the arguments can be packed into a structure, and the address of that structure can be passed to <i>arg</i>.</p> <p>If <i>start_func</i> returns, the thread will terminate with the exit status set to the <i>start_func</i> return value (see thr_exit(3T)).</p> <p>When the thread returns in which main() originated from, the effect is the same as if there were an implicit call to exit() using the return value of main() as the exit status. This differs from a <i>start_func</i> return. However, if main() itself calls thr_exit(3T), only the main thread exits, not the entire process.</p> <p>If the thread creation itself fails, a new thread is not created and the contents of the location referenced by the pointer to the new thread are undefined.</p> |

flags specifies which attributes are modifiable for the created thread. The value in *flags* is determined by the bitwise inclusive OR of the following:

| | |
|---------------|--|
| THR_BOUND | This flag affects the contentscope attribute of the thread. The new thread is created permanently bound to an LWP (that is, it is a <i>bound thread</i>). This thread will now contend among system-wide resources. |
| THR_DETACHED | This flag affects the detachstate attribute of the thread. The new thread is created detached. The exit status of a detached thread is not accessible to other threads. Its thread ID and other resources may be re-used as soon as the thread terminates. <code>thr_join(3T)</code> will not wait for a detached thread. |
| THR_NEW_LWP | This flag affects the concurrency attribute of the thread. The desired concurrency level for unbound threads is increased by one. This is similar to incrementing concurrency by one by way of <code>thr_setconcurrency(3T)</code> . Typically, this adds a new LWP to the pool of LWPs running unbound threads. |
| THR_SUSPENDED | This flag affects the suspended attribute of the thread. The new thread is created suspended and will not execute <i>start_func</i> until it is started by <code>thr_continue()</code> . |
| THR_DAEMON | This flag affects the daemon attribute of the thread. The thread is marked as a daemon. The process will exit when all non-daemon threads exit. <code>thr_join(3T)</code> will not wait for a daemon thread. Daemon threads do not interfere with the exit conditions for a process. A process will terminate when all regular threads exit or the process calls <code>exit()</code> . Daemon threads are most useful in libraries that want to use threads. |

Default thread creation:

```
thread_t tid;
void *start_func(void *), *arg;
```

```
thr_create(NULL, NULL, start_func, arg, NULL, &tid);
```

User-defined thread creation: create a thread scheduled on a system-wide basis (that is, a bound thread), use:

```
thr_create(NULL, NULL, start_func, arg, THR_BOUND, &tid);
```

Another example of customization is, if both `THR_BOUND` and `THR_NEW_LWP` are specified then, typically, two LWPs are created, one for the bound thread and another for the pool of LWPs running unbound threads.

```
thr_create(NULL, NULL, start_func, arg, THR_BOUND | THR_NEW_LWP, &tid);
```

With `thr_create()`, the new thread will use the stack starting at the address specified by `stack_base` and continuing for `stack_size` bytes. `stack_size` must be greater than the value returned by `thr_min_stack(3T)`. If `stack_base` is `NULL` then `thr_create()` allocates a stack for the new thread with at least `stack_size` bytes. If `stack_size` is 0 then a default size is used. If `stack_size` is not 0 then it must be greater than the value returned by `thr_min_stack(3T)`. See NOTES.

When `new_thread_ID` is not `NULL` then it points to a location where the ID of the new thread is stored if `thr_create()` is successful. The ID is only valid within the calling process.

RETURN VALUES

If successful, the `thr_create()` function returns 0. Otherwise, an error number is returned to indicate the error.

ERRORS

The `thr_create()` function will fail if:

- EAGAIN** The system-imposed limit on the total number of threads in a process has been exceeded or some system resource has been exceeded (for example, too many LWPs were created).
- EINVAL**
- `stack_base` is not `NULL` and `stack_size` is less than the value returned by `thr_min_stack(3T)`
 - `stack_base` is `NULL` and `stack_size` is not 0 and is less than the value returned by `thr_min_stack(3T)`

EXAMPLES

EXAMPLE 1 This is an example of concurrency with multi-threading. Since POSIX threads and Solaris threads are fully compatible even within the same process, this example uses `pthread_create()` if you execute `a.out 0`, or `thr_create()` if you execute `a.out 1`.

Five threads are created that simultaneously perform a time-consuming function, `sleep(10)`. If the execution of this process is timed, the results will show that all five individual calls to sleep for ten-seconds completed in about ten seconds, even on a uniprocessor. If a single-threaded process calls `sleep(10)` five times, the execution time will be about 50-seconds.

The command-line to time this process is:

```
/usr/bin/time a.out 0 (for POSIX threading)
```

or

```
/usr/bin/time a.out 1 (for Solaris threading)
```

```
/* cc thisfile.c -lthread -lpthread */
#define _REENTRANT /* basic 3-lines for threads */
#include <pthread.h>
#include <thread.h>
#define NUM_THREADS 5
#define SLEEP_TIME 10

void *sleeping(void *); /* thread routine */
int i;
thread_t tid[NUM_THREADS]; /* array of thread IDs */

int
main(int argc, char *argv[])
{
    if (argc == 1) {
        printf("use 0 as arg1 to use pthread_create()\n");
        printf("or use 1 as arg1 to use thr_create()\n");
        return (1);
    }

    switch (*argv[1]) {
    case '0': /* POSIX */
        for ( i = 0; i < NUM_THREADS; i++)
            pthread_create(&tid[i], NULL, sleeping,
                (void *)SLEEP_TIME);
        for ( i = 0; i < NUM_THREADS; i++)
            pthread_join(tid[i], NULL);
        break;

    case '1': /* Solaris */
        for ( i = 0; i < NUM_THREADS; i++)
            thr_create(NULL, 0, sleeping, (void *)SLEEP_TIME, 0,
                &tid[i]);
        while (thr_join(NULL, NULL, NULL) == 0)
            ;
        break;
    } /* switch */
}
```

```

        printf("main() reporting that all %d threads have terminated\n", i);
        return (0);
    } /* main */

    void *
    sleeping(void *arg)
    {
        int sleep_time = (int)arg;
        printf("thread %d sleeping %d seconds ...\n", thr_self(), sleep_time);
        sleep(sleep_time);
        printf("\nthread %d awakening\n", thr_self());
        return (NULL);
    }

```

If **main()** had not waited for the completion of the other threads (using **pthread_join(3T)** or **thr_join(3T)**), it would have continued to process concurrently until it reached the end of its routine and the entire process would have exited prematurely (see **exit(2)**).

EXAMPLE 2 Creating a default thread with a new signal mask.

The following example shows how to create a default thread with a new signal mask. **new_mask** is assumed to have a different value than the creator's signal mask (**orig_mask**). **new_mask** is set to block all signals except for **SIGINT**. The creator's signal mask is changed so that the new thread inherits a different mask, and is restored to its original value after **thr_create()** returns.

This example assumes that **SIGINT** is also unmasked in the creator. If it is masked by the creator, then unmasking the signal opens the creator up to this signal. The other alternative is to have the new thread set its own signal mask in its start routine.

```

thread_t tid;
sigset_t new_mask, orig_mask;
int error;

(void) sigfillset(&new_mask);
(void) sigdelset(&new_mask, SIGINT);
(void) thr_sigsetmask(SIG_SETMASK, &new_mask, &orig_mask);
error = thr_create(NULL, 0, do_func, NULL, 0, &tid);
(void) thr_sigsetmask(SIG_SETMASK, &orig_mask, NULL);

```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`_lwp_create(2)`, `exit(2)`, `exit(3C)`, `sleep(3C)`, `thr_min_stack(3T)`, `thr_setconcurrency(3T)`, `thr_suspend(3T)`, `threads(3T)`, `attributes(5)`, `standards(5)`

NOTES

MT application threads execute independently of each other, thus their relative behavior is unpredictable. Therefore, it is possible for the thread executing `main()` to finish before all other user application threads.

Using `thr_join(3T)` in the following syntax,

```
while (thr_join(NULL, NULL, NULL) == 0);
```

will cause the invoking thread (which may be `main()`) to wait for the termination of all other undetached and non-daemon threads; however, the second and third arguments to `thr_join(3T)` need not necessarily be `NULL`.

A thread has not terminated until `thr_exit()` has finished. The only way to determine this is by `thr_join()`. When `thr_join()` returns a departed thread, it means that this thread has terminated and its resources are reclaimable. For instance, if a user specified a stack to `thr_create()`, this stack can only be reclaimed after `thr_join()` has reported this thread as a departed thread. It is not possible to determine when a *detached* thread has terminated. A detached thread disappears without leaving a trace.

Typically, thread stacks allocated by `thr_create()` begin on page boundaries and any specified (a red-zone) size is rounded up to the next page boundary. A page with no access permission is appended to the top of the stack so that most stack overflows will result in a `SIGSEGV` signal being sent to the offending thread. Thread stacks allocated by the caller are used as is.

Using a default stack size for the new thread, instead of passing a user-specified stack size, results in much better `thr_create()` performance. The default stack size for a user-thread is 1 megabyte in a 32-bit process and 2 megabyte in a 64-bit process.

A user-specified stack size must be greater than the value `THR_MIN_STACK`. A minimum stack size may not accommodate the stack frame for the user thread function *start_func*. If a stack size is specified, it must accommodate *start_func* requirements and the functions that it may call in turn, in addition to the minimum requirement.

It is usually very difficult to determine the runtime stack requirements for a thread. `THR_MIN_STACK` specifies how much stack storage is required to execute a `NULL start_func`. The total runtime requirements for stack storage are dependent on the storage required to do runtime linking, the amount of storage required by library runtimes (like `printf()`) that your thread calls. Since these storage parameters are not known before the program runs, it is best to use

default stacks. If you know your runtime requirements or decide to use stacks that are larger than the default, then it makes sense to specify your own stacks.

| | |
|---------------------|---|
| NAME | threads, pthreads, libpthread, libthread – concepts related to POSIX pthreads and Solaris threads and the libpthread and libthread libraries |
| SYNOPSIS | |
| POSIX | <pre>cc - mt [<i>flag</i> ...] <i>file</i> ...- lpthread [-lposix4 <i>library</i> ...] #include <pthread.h></pre> |
| Solaris | <pre>cc - mt [<i>flag</i> ...] <i>file</i> ...[<i>library</i> ...] #include <sched.h> #include <thread.h></pre> |
| DESCRIPTION | <p>POSIX and Solaris threads each have their own implementation of the threads library. The <code>libpthread</code> library is associated with POSIX; the <code>libthread</code> library is associated with Solaris. Both implementations are interoperable, their functionality similar, and can be used within the same application. Only POSIX threads are guaranteed to be fully portable to other POSIX-compliant environments. POSIX and Solaris threads require different source, include files and linking libraries. See <code>SYNOPSIS</code> .</p> |
| Similarities | <p>Most of the functions in the <code>libpthread</code> and <code>libthread</code> , libraries have a counterpart in the other corresponding library. POSIX function names, with the exception of the semaphore names, have a " <code>pthread</code> " prefix. Function names for similar POSIX and Solaris have similar endings. Typically, similar POSIX and Solaris functions have the same number and use of arguments.</p> |
| Differences | <p>POSIX pthreads and Solaris threads differ in the following ways:</p> <ul style="list-style-type: none"> ■ POSIX threads are more portable. ■ POSIX threads establish characteristics for each thread according to configurable attribute objects. ■ POSIX pthreads implement thread cancellation. ■ POSIX pthreads enforce scheduling algorithms. ■ POSIX pthreads allow for clean-up handlers for <code>fork(2)</code> calls. ■ Solaris threads can be suspended and continued. ■ Solaris threads implement an optimized mutex and interprocess robust mutex locks. ■ Solaris threads implement daemon threads, for whose demise the process does not wait. |

**Function
Comparison**

The following table compares the POSIX pthreads and Solaris threads functions. When a comparable interface is not available either in POSIX pthreads or Solaris threads, a hyphen (--) appears in the column.

**Functions Related to
Creation**

| POSIX (libpthread) | Solaris (libthread) |
|--------------------------------|---------------------|
| pthread_create() | thr_create() |
| pthread_attr_init() | -- |
| pthread_attr_setdetachstate() | -- |
| pthread_attr_getdetachstate() | -- |
| pthread_attr_setinheritsched() | -- |
| pthread_attr_getinheritsched() | -- |
| pthread_attr_setschedparam() | -- |
| pthread_attr_getschedparam() | -- |
| pthread_attr_setschedpolicy() | -- |
| pthread_attr_getschedpolicy() | -- |
| pthread_attr_setscope() | -- |
| pthread_attr_getscope() | -- |
| pthread_attr_setstackaddr() | -- |
| pthread_attr_getstackaddr() | -- |
| pthread_attr_setstacksize() | -- |
| pthread_attr_getstacksize() | -- |
| pthread_attr_getguardsize() | -- |
| pthread_attr_setguardsize() | -- |
| pthread_attr_destroy() | -- |
| -- | thr_min_stack() |

**Functions Related to
Exit**

| POSIX (libpthread) | Solaris (libthread) |
|--------------------|---------------------|
| pthread_exit() | thr_exit() |
| pthread_join() | thr_join() |
| pthread_detach() | -- |

Functions Related to Thread Specific Data

| POSIX (libpthread) | Solaris (libthread) |
|-----------------------|---------------------|
| pthread_key_create() | thr_keycreate() |
| pthread_setspecific() | thr_setspecific() |
| pthread_getspecific() | thr_getspecific() |
| pthread_key_delete() | -- |

Functions Related to Signals

| POSIX (libpthread) | Solaris (libthread) |
|--------------------|---------------------|
| pthread_sigmask() | thr_sigsetmask() |
| pthread_kill() | thr_kill() |

Functions Related to IDs

| POSIX (libpthread) | Solaris (libthread) |
|--------------------|---------------------|
| pthread_self() | thr_self() |
| pthread_equal() | -- |
| -- | thr_main() |

Functions Related to Scheduling

| POSIX (libpthread) | Solaris (libthread) |
|--------------------------|----------------------|
| -- | thr_yield() |
| -- | thr_suspend() |
| -- | thr_continue() |
| pthread_setconcurrency() | thr_setconcurrency() |
| pthread_getconcurrency() | thr_getconcurrency() |
| pthread_setschedparam() | thr_setprio() |
| pthread_getschedparam() | thr_getprio() |

Functions Related to Cancellation

| POSIX (libpthread) | Solaris (libthread) |
|--------------------------|---------------------|
| pthread_cancel() | -- |
| pthread_setcancelstate() | -- |
| pthread_setcanceltype() | -- |
| pthread_testcancel() | -- |
| pthread_cleanup_pop() | -- |
| pthread_cleanup_push() | -- |

Functions Related to Mutexes

| POSIX (libpthread) | Solaris (libthread) |
|------------------------------------|---------------------|
| pthread_mutex_init() | mutex_init() |
| pthread_mutexattr_init() | -- |
| pthread_mutexattr_setpshared() | -- |
| pthread_mutexattr_getpshared() | -- |
| pthread_mutexattr_setprotocol() | -- |
| pthread_mutexattr_getprotocol() | -- |
| pthread_mutexattr_setprioceiling() | -- |
| pthread_mutexattr_getprioceiling() | -- |
| pthread_mutexattr_settype() | -- |
| pthread_mutexattr_gettype() | -- |
| pthread_mutexattr_destroy() | -- |
| pthread_mutex_setprioceiling() | -- |
| pthread_mutex_getprioceiling() | -- |
| pthread_mutex_lock() | mutex_lock() |
| pthread_mutex_trylock() | mutex_trylock() |
| pthread_mutex_unlock() | mutex_unlock() |
| pthread_mutex_destroy() | mutex_destroy() |

**Functions Related to
Condition Variables**

| POSIX (libpthread) | Solaris (libthread) |
|-------------------------------|---------------------|
| pthread_cond_init() | cond_init() |
| pthread_condattr_init() | -- |
| pthread_condattr_setpshared() | -- |
| pthread_condattr_getpshared() | -- |
| pthread_condattr_destroy() | -- |
| pthread_cond_wait() | cond_wait() |
| pthread_cond_timedwait() | cond_timedwait() |
| pthread_cond_signal() | cond_signal() |
| pthread_cond_broadcast() | cond_broadcast() |
| pthread_cond_destroy() | cond_destroy() |

**Functions Related to
Reader/Writer
Locking**

| POSIX (libpthread) | Solaris (libthread) |
|---------------------------------|---------------------|
| pthread_rwlock_init() | rwlock_init() |
| pthread_rwlock_rdlock() | rw_rdlock() |
| pthread_rwlock_tryrdlock() | rw_tryrdlock() |
| pthread_rwlock_wrlock() | rw_wrlock() |
| pthread_rwlock_trywrlock() | rw_trywrlock() |
| pthread_rwlock_unlock() | rw_unlock() |
| pthread_rwlock_destroy() | rwlock_destroy() |
| pthread_rwlockattr_init() | -- |
| pthread_rwlockattr_destroy() | -- |
| pthread_rwlockattr_getpshared() | -- |
| pthread_rwlockattr_setpshared() | -- |

**Functions Related to
Semaphores**

| POSIX (libpthread) | Solaris (libthread) |
|--------------------|---------------------|
| sem_init() | sema_init() |
| sem_open() | -- |

| | |
|----------------|----------------|
| sem_close() | -- |
| sem_wait() | sema_wait() |
| sem_trywait() | sema_trywait() |
| sem_post() | sema_post() |
| sem_getvalue() | -- |
| sem_unlink() | -- |
| sem_destroy() | sema_destroy() |

Functions Related to fork() Clean Up

| | |
|--------------------|---------------------|
| POSIX (libpthread) | Solaris (libthread) |
| pthread_atfork() | -- |

Functions Related to Limits

| | |
|--------------------|---------------------|
| POSIX (libpthread) | Solaris (libthread) |
| pthread_once() | -- |

Functions Related to Debugging

| | |
|--------------------|---------------------|
| POSIX (libpthread) | Solaris (libthread) |
| -- | thr_stksegment() |

LOCKING

Synchronization

POSIX (libpthread)\011Solaris (libthread) Multi-threaded behavior is asynchronous, and therefore, optimized for concurrent and parallel processing. As threads, always from within the same process and sometimes from multiple processes, share global data with each other, they are not guaranteed exclusive access to the shared data at any point in time. Securing mutually exclusive access to shared data requires synchronization among the threads. Both POSIX and Solaris implement four synchronization mechanisms: mutexes, condition variables, reader/writer locking (*optimized frequent-read occasional-write mutex*), and semaphores.

Synchronizing multiple threads diminishes their concurrency. The coarser the grain of synchronization, that is, the larger the block of code that is locked, the lesser the concurrency.

MT fork() If a POSIX threads program calls `fork(2)`, it implicitly calls `fork1(2)`, which replicates only the calling thread. Should there be any outstanding mutexes throughout the process, the application should call `pthread_atfork(3T)`, to wait for and acquire those mutexes, prior to calling `fork()`.

SCHEDULING

POSIX Scheduling allocation size per thread is greater than one. POSIX supports the following three scheduling policies:

`SCHED_OTHER` Timesharing (TS) scheduling policy. It is based on the timesharing scheduling class.

`SCHED_FIFO` First-In-First-Out (FIFO) scheduling policy. Threads scheduled to this policy, if not pre-empted by a higher priority, will proceed until completion. Threads whose contention scope is system (`PTHREAD_SCOPE_SYSTEM`) are in real-time (RT) scheduling class. The calling process must have a effective user ID of 0. `SCHED_FIFO` for threads whose contention scope's process (`PTHREAD_SCOPE_PROCESS`) is based on the TS scheduling class.

`SCHED_RR` Round-Robin scheduling policy. Threads scheduled to this policy, if not pre-empted by a higher priority, will execute for a time period determined by the system. Threads whose contention scope is system (`PTHREAD_SCOPE_SYSTEM`) are in real-time (RT) scheduling class and the calling process must have a effective user ID of 0. `SCHED_RR` for threads whose contention scope is process (`PTHREAD_SCOPE_PROCESS`) is based on the TS scheduling class.

Solaris Only scheduling policy supported is `SCHED_OTHER`, which is timesharing, based on the TS scheduling class.

ERRORS In a multi-threaded application, linked with `libpthread` or `libthread`, `EINTR` may be returned whenever another thread calls `fork(2)`, which calls `fork1(2)` instead.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|----------------------|
| MT-Level | MT-Safe, Fork 1-Safe |

FILES

POSIX /usr/include/pthread.h /lib/libpthread.* /lib/libposix4.*

Solaris /usr/include/thread.h /usr/include/sched.h /lib/libthread.*

SEE ALSO `fork(2)`, `pthread_atfork(3T)`, `pthread_create(3T)`, `attributes(5)`,
`standards(5)`

| | |
|--------------------|---|
| NAME | thr_exit – terminate the calling thread |
| SYNOPSIS | <pre>cc - mt [flag...] file...[library...] #include <thread.h> void thr_exit(void *status);</pre> |
| DESCRIPTION | <p>thr_exit() terminates the calling thread, in a similar way that exit(3C) terminates the calling process. If the calling thread is not detached, then the thread's ID and the exit status specified by <i>status</i> are retained. The value <i>status</i> is then made available to any successful join with the terminating thread (see thr_join(3T)); otherwise, <i>status</i> is disregarded allowing the thread's ID to be reclaimed immediately.</p> <p>Any cancellation cleanup handlers that have been pushed and not yet popped are popped in the reverse order that they were pushed and then executed. After all cancellation cleanup handlers have been executed, if the thread has any thread-specific data, appropriate destructor functions will be called in an unspecified order. Thread termination does not release any application visible process resources, including, but not limited to, mutexes and file descriptors, nor does it perform any process level cleanup actions, including, but not limited to, calling any atexit() routines that may exist.</p> <p>If any thread, including the main() thread, calls thr_exit(), only that thread will exit.</p> <p>If main() returns or exits (either implicitly or explicitly), or any thread explicitly calls exit(), the entire process will exit.</p> <p>The behavior of thr_exit() is undefined if called from a cancellation cleanup handler or destructor function that was invoked as a result of either an implicit or explicit call to thr_exit().</p> <p>After a thread has terminated, the result of access to local (auto) variables of the thread is undefined. Thus, references to local variables of the exiting thread should not be used for the thr_exit() <i>status</i> parameter value.</p> <p>The process exits with an exit status of 0 after the last thread has been terminated. The behavior is as if the implementation called exit() with a 0 argument at thread termination time.</p> <p>If any thread (except the main() thread) implicitly or explicitly returns, the result is the same as if the thread called thr_exit() and it will return the value of <i>status</i> as the exit code.</p> <p>The process will terminate with an exit status of 0 after the last thread has terminated (including the main() thread). This action is the same as if the application had called exit() with a 0 argument at thread termination time.</p> |

RETURN VALUES

The **thr_exit()** function cannot return to its caller.

ERRORS

No errors are defined.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

exit(3C), **thr_create(3T)**, **thr_join(3T)**, **thr_keycreate(3T)**,
attributes(5), **standards(5)**

NOTES

Although only POSIX implements cancellation, cancellation can be used with Solaris threads, due to their interoperability.

status should not reference any variables local to the calling thread.

| | |
|----------------------|---|
| NAME | thr_getconcurrency, thr_setconcurrency – get or set thread concurrency level |
| SYNOPSIS | <pre>cc - mt [<i>flag</i> ...] <i>file</i> ...[<i>library</i> ...] #include <thread.h> int thr_setconcurrency(int <i>new_level</i>); int thr_getconcurrency(void);</pre> |
| DESCRIPTION | <p>Unbound threads in a process may or may not be required to be simultaneously active. See thr_create(3T) . By default, the threads system ensures that a sufficient number of threads are active so that the process can continue to make progress. While this conserves system resources, it may not produce the most effective level of concurrency. thr_setconcurrency() permits the application to give the threads system a hint, specified by <i>new_level</i> , for the desired level of concurrency. The actual number of simultaneously active threads may be larger or smaller than this number. The value for the desired concurrency level may also be affected by creating threads with the THR_NEW_LWP flag set. See thr_create(3T) .</p> <p>If <i>new_level</i> is 0 , the threads system will only ensure that a sufficient number of threads are active so that the process can continue to make progress.</p> <p>thr_getconcurrency() returns the current value for the desired concurrency level. The actual number of simultaneously active threads may be larger or smaller than this number.</p> |
| RETURN VALUES | <p>The thr_getconcurrency() function always returns the current value for the desired concurrency level.</p> <p>If successful, the thr_setconcurrency() function returns 0 . Otherwise, a non-zero value is returned to indicate the error.</p> |
| ERRORS | <p>The thr_setconcurrency() function will fail if:</p> <p>EAGAIN The specified concurrency level would cause a system resource to be exceeded.</p> <p>EINVAL <i>new_level</i> is negative.</p> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO`thr_create(3T)`, `attributes(5)`, `standards(5)`

| | |
|------------------------|---|
| NAME | thr_getprio, thr_setprio – access dynamic thread scheduling |
| SYNOPSIS | <pre>cc - mt [<i>flag</i> ...] <i>file</i> ...[<i>library</i> ...] #include <thread.h> int thr_setprio(thread_t target_thread, int priority); int thr_getprio(thread_t target_thread, int * priority);</pre> |
| DESCRIPTION | <p>Thread scheduling is controlled by three attributes: its scope of contention, being either inter-process or intra-process (bound vs. unbound), (see <code>prionctl(2)</code>); a relative scheduling priority; and a scheduling policy.</p> |
| Contentionscope | <p>Bound threads, which are inter-process, compete system-wide for scheduling resources and must be set at creation, for example:</p> <pre>thr_create(NULL, NULL, thread_routine, arg, THR_BOUND, NULL);</pre> <p>A bound thread is bound to an LWP and its scheduling is dependent upon the scheduling of the LWP to which it is bound. LWPs compete with other LWPs in other processes, however, their scheduling may be dynamically controlled by <code>prionctl(2)</code>.</p> <p>By default, the scope for newly-created threads are unbound, or intra-process, and their setting is <code>NULL</code>. An unbound thread is scheduled by <code>libthread</code> on an underlying LWP, which competes with other LWPs in the same process.</p> <p>The following dynamic scheduling functions should be used only with unbound threads: <code>thr_setprio()</code>, and <code>thr_getprio()</code>.</p> |
| Priority | <p>Priority scheduling is determined as follows:</p> <ul style="list-style-type: none"> ■ Higher priority threads are scheduled before lower priority threads. ■ Solaris threads assumes that the priority is inherited across a thread create. ■ A Solaris thread can be created suspended and its priority can be modified. |

thr_setprio() can dynamically modify an unbound thread's priority, and **thr_getprio()** can read an unbound thread's priority.

Policy

The scheduling *policy* setting is:

SCHED_OTHER (*system default, often time-sharing*) Competing threads in this class are multiplexed according to their relative *priority*.

Scheduling

Solaris scheduling may only dynamically affect *priority*. There is no functionality to alter the *policy* of any thread; by default, a Solaris thread's schedule is equivalent to **SCHED_OTHER**, which is the only available Solaris policy.

thr_setprio() changes the priority of the thread, specified by *target_thread*, within the current process to the priority specified by *priority*. Currently, by default, threads are scheduled based on fixed priorities that range from zero, the least significant, to 127. The *target_thread* will preempt lower priority threads, and will yield to higher priority threads in their contention for LWPs, not CPUs.

The function **thr_getprio()** stores the current priority for the thread specified by *target_thread* in the location pointed to by *priority*. Note that thread priorities regulate access to LWPs, not CPUs, and hence are different from real-time priorities, which regulate and enforce access to CPU resources. A thread's priority set via these functions is more like a hint in terms of guaranteed access to execution resources. Programs that need access to "real" priorities should use bound threads in the real-time class (see **pricnt1(2)**).

RETURN VALUES

If successful, the **thr_getprio()** and **thr_setprio()** return 0. Otherwise, an error number is returned to indicate the error.

ERRORS

For each of the following conditions, these functions return an error number if the condition is detected.

ESRCH The value specified by *target_thread* does not refer to an existing thread.

The **thr_getprio()** and **thr_setprio()** functions may fail if:

EINVAL The value of *priority* makes no sense for the scheduling class associated with the *target_thread*.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

thr_getprio(3T)

Threads Library

SEE ALSO

priocntl(2), sched_setparam(3R), thr_create(3T),
thr_suspend(3T), thr_yield(3T), attributes(5), standards(5)

| NAME | thr_join – wait for thread termination | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc - mt [flag...] file...[library...] #include <thread.h> int thr_join(thread_t thread, thread_t *departed, void **status);</pre> | | | | |
| DESCRIPTION | <p>The thr_join() functions suspend processing of the calling thread until the target <i>thread</i> completes. <i>thread</i> must be a member of the current process and it cannot be a detached or daemon thread. See thr_create(3T).</p> <p>Several threads cannot wait for the same thread to complete; one thread will complete successfully and the others will terminate with an error of ESRCH. thr_join() will not block processing of the calling thread if the target <i>thread</i> has already terminated.</p> <p>thr_join() returns successfully when the target <i>thread</i> terminates.</p> <p>If a thr_join() call returns successfully with a non-null <i>status</i> argument, the value passed to thr_exit(3T) by the terminating thread will be placed in the location referenced by <i>status</i>.</p> <p>If the target <i>thread</i> ID is 0, thr_join() waits for any undetached thread in the process to terminate.</p> <p>If <i>departed</i> is not NULL, it points to a location that is set to the ID of the terminated thread if thr_join() returns successfully.</p> | | | | |
| RETURN VALUES | If successful, thr_join() returns 0. Otherwise, an error number is returned to indicate the error. | | | | |
| ERRORS | <p>ESRCH No undetached thread could be found corresponding to that specified by the given thread ID.</p> <p>EDEADLK A recursive deadlock was detected, the value of <i>thread</i> specifies the calling thread. See NOTES.</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | wait(2) , thr_create(3T) , thr_exit(3T) , attributes(5) , standards(5) | | | | |

NOTES

Using `thr_join(3T)` in the following syntax,

```
while (thr_join(NULL, NULL, NULL) == 0);
```

will wait for the termination of all other undetached and non-daemon threads; after which, EDEADLK will be returned.

| | |
|--------------------|---|
| NAME | thr_keycreate, thr_setspecific, thr_getspecific – thread-specific-data functions |
| SYNOPSIS | <pre>cc - mt [<i>flag</i> ...] <i>file</i> ...[<i>library</i> ...] #include <thread.h> int thr_keycreate(thread_key_t * <i>keyp</i>, void (* <i>destructor</i>, void * <i>value</i>); int thr_setspecific(thread_key_t <i>key</i>, void * <i>value</i>); int thr_getspecific(thread_key_t <i>key</i>, void ** <i>valuep</i>);</pre> |
| DESCRIPTION | |
| Create Key | <p>In general, thread key creation allocates a key that locates data specific to each thread in the process. The key is global to all threads in the process, which allows each thread to bind a value to the key once the key has been created. The key independently maintains specific values for each binding thread. The thr_keycreate() function allocates a global <i>key</i> namespace, pointed to by <i>keyp</i>, that is visible to all threads in the process. Each thread is initially bound to a private element of this <i>key</i>, which allows access to its thread-specific data.</p> <p>Upon key creation, a new key is assigned the value <code>NULL</code> for all active threads. Additionally, upon thread creation, all previously created keys in the new thread are assigned the value <code>NULL</code>.</p> <p>Optionally, a destructor function, <i>destructor</i>, may be associated with each <i>key</i>. Upon thread exit, if a <i>key</i> has a non-<code>NULL</code> <i>destructor</i> function and the thread has a non-<code>NULL</code> <i>value</i> associated with that <i>key</i>, the <i>destructor</i> function is called with the current associated <i>value</i>. If more than one <i>destructor</i> exists for a thread when it exits, the order of destructor calls is unspecified.</p> |
| Set Value | <p>Once a key has been created, each thread may bind a new <i>value</i> to the key using thr_setspecific(). The values are unique to the binding thread and are individually maintained. These values continue for the life of the calling thread.</p> <p>Proper synchronization of <i>key</i> storage and access must be ensured by the caller. The <i>value</i> argument to thr_setspecific() is generally a pointer to a block of dynamically allocated memory reserved by the calling thread for its own use. See <code>EXAMPLES</code>.</p> |

At thread exit, the *destructor* function, which is associated at time of creation, is called and it uses the specific key value as its sole argument.

Get Value **thr_getspecific()** stores the current value bound to *key* for the calling thread into the location pointed to by *valuep* .

RETURN VALUES If successful, **thr_keycreate()** , **thr_setspecific()** and **thr_getspecific()** return 0 . Otherwise, an error number is returned to indicate the error.

ERRORS If the following conditions occur, **thr_keycreate()** returns the corresponding error number:

EAGAIN The system lacked the necessary resources to create another thread-specific data key.

ENOMEM Insufficient memory exists to create the key.

If the following conditions occur, **thr_keycreate()** and **thr_setspecific()** return the corresponding error number:

ENOMEM Insufficient memory exists to associate the value with the key.

The **thr_setspecific()** function returns the corresponding error number:

EINVAL The *key* value is invalid.

EXAMPLES

EXAMPLE 1 In this example, the thread-specific data in this function can be called from more than one thread without special initialization.

For each argument you pass to the executable of this example, a thread is created and privately bound to the string-value of that argument.

```

/* cc thisfile.c */

#define _REENTRANT
#include <thread.h>
void *thread_specific_data(), free();
#define MAX_ARGC 20
thread_t tid[MAX_ARGC];
int num_threads;

main( int argc, char *argv[] ) {
    int i;
    num_threads = argc - 1;
    for( i = 0; i < num_threads; i++)
        thr_create(NULL, 0, thread_specific_data, argv[i+1]);
    for( i = 0; i < num_threads; i++)
        thr_join(tid[i], NULL, NULL);
} /* end main */

void *thread_specific_data(char private_data[])
{
    static mutex_tkeylock; /* static ensures only one copy of keylock */

```

```

static thread_key_tkey;
static intonce_per_keyname = 0;
void *tsd = NULL;

if (!once_per_keyname) {
    mutex_lock(&keylock);
    if (!once_per_keyname) {
        thr_keycreate(&key, free);
        once_per_keyname++;
    }
    mutex_unlock(&keylock);
}
tsd = thr_getspecific(key);
if (tsd == NULL) {
    tsd = (void *)malloc(strlen(private_data) + 1);
    strcpy(tsd, private_data);
    thr_setspecific(key, tsd);
    printf("tsd for %d = %s\n", thr_self(), (char *)thr_getspecific(key));
    sleep(2);
    printf("tsd for %d remain\n", thr_self(), (char *)thr_getspecific(key));
} /* end thread_specific_data */

void
free(void *v) {
    /* application-specific clean-up function */
}

```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

thr_exit(3T), **attributes(5)**, **standards(5)**

WARNINGS

The **thr_getspecific()** and **thr_setspecific()** functions may be called either explicitly, or implicitly from a thread-specific data destructor function. Calling **thr_setspecific()** from a destructor may result in lost storage or infinite loops.

| NAME | thr_kill – send a signal to a thread | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc - mt [flag...] file...[library...] #include <signal.h> #include <thread.h> int thr_kill(thread_t thread, int sig);</pre> | | | | |
| DESCRIPTION | <p>thr_kill() sends the <i>sig</i> signal to the thread designated by <i>thread</i>. <i>thread</i> must be a member of the same process as the calling thread. <i>sig</i> must be one of the signals listed in signal(5); with the exception of SIGLWP, SIGCANCEL, and SIGWAITING being reserved and off limits to thr_kill(). If <i>sig</i> is 0, a validity check is done for the existence of the target thread; no signal is sent.</p> | | | | |
| RETURN VALUES | <p>Upon successful completion, thr_kill() returns 0. Otherwise, an error number is returned. In the event of failure, no signal is sent.</p> | | | | |
| ERRORS | <p>ESRCH No thread was found that corresponded to the thread designated by <i>thread</i> ID.</p> <p>EINVAL The <i>sig</i> argument value is not zero and is an invalid or an unsupported signal number.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | <p>kill(2), sigaction(2), raise(3C), thr_self(3T), attributes(5), signal(5), standards(5)</p> | | | | |

NAME thr_main - identify the main thread

SYNOPSIS

```
cc - mt [ flag... ] file...[ library... ]

#include <thread.h>

int thr_main(void);
```

DESCRIPTION The **thr_main()** function returns one of the following:

- 1 if the calling thread is the main thread
- 0 if the calling thread is not the main thread
- 1 if **libthread** is not linked in or thread initialization has not completed

FILES

/lib/libthread

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **thr_self(3T)**, **attributes(5)**

| | |
|--------------------|---|
| NAME | thr_min_stack – return the minimum-allowable size for a thread's stack |
| SYNOPSIS | <pre>cc - mt [flag...] file...[library...] #include <thread.h> size_t thr_min_stack(void);</pre> |
| DESCRIPTION | <p>When a thread is created with a user-supplied stack, the user must reserve enough space to run this thread. In a dynamically linked execution environment, it is very hard to know what the minimum stack requirements are for a thread. The function thr_min_stack() returns the amount of space needed to execute a null thread. This is a thread that was created to execute a null procedure. A thread that does something useful should have a stack size that is thr_min_stack() + <some increment>.</p> <p>Most users should not be creating threads with user-supplied stacks. This functionality was provided to support applications that wanted complete control over their execution environment.</p> <p>Typically, users should let the threads library manage stack allocation. The threads library provides default stacks which should meet the requirements of any created thread.</p> <p>thr_min_stack() will return the unsigned int <code>THR_MIN_STACK</code>, which is the minimum-allowable size for a thread's stack.</p> <p>In this implementation the default size for a user-thread's stack is one mega-byte. If the second argument to thr_create(3T) is <code>NULL</code>, then the default stack size for the newly-created thread will be used. Otherwise, you may specify a stack-size that is at least <code>THR_MIN_STACK</code>, yet less than the size of your machine's virtual memory.</p> <p>It is recommended that the default stack size be used.</p> <p>To determine the smallest-allowable size for a thread's stack, execute the following:</p> <pre>/* cc thisfile.c -lthread */ #define _REENTRANT #include <thread.h> #include <stdio.h> main() { printf("thr_min_stack() returns %u\n",thr_min_stack()); }</pre> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO `attributes(5)`, `standards(5)`

NAME thr_self – get calling thread's ID

SYNOPSIS cc - mt [*flag...*] *file...*[*library...*]
#include <thread.h>

```
thread_t thr_self(void);
typedef(unsigned int thread_t);
```

DESCRIPTION **thr_self()** returns the thread ID of the calling thread.

ERRORS No errors are defined.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **thr_create(3T)**, **attributes(5)**, **standards(5)**

| | |
|----------------------|--|
| NAME | thr_sigsetmask – change or examine calling thread’s signal mask |
| SYNOPSIS | <pre>cc - mt [<i>flag...</i>] <i>file...</i>[<i>library...</i>] #include <thread.h> #include <signal.h> int thr_sigsetmask(int <i>how</i>, const sigset_t *set, sigset_t *oset);</pre> |
| DESCRIPTION | <p>The thr_sigsetmask() function changes or examines a calling thread’s signal mask. Each thread has its own signal mask. A new thread inherits the calling thread’s signal mask and priority; however, pending signals are not inherited. Signals pending for a new thread will be empty.</p> <p>If the value of the argument <i>set</i> is not <code>NULL</code>, <i>set</i> points to a set of signals that can modify the currently blocked set. If the value of <i>set</i> is <code>NULL</code>, the value of <i>how</i> is insignificant and the thread’s signal mask is unmodified; thus, thr_sigsetmask() can be used to inquire about the currently blocked signals.</p> <p>The value of the argument <i>how</i> specifies the method in which the set is changed and takes one of the following values:</p> <p><code>SIG_BLOCK</code> <i>set</i> corresponds to a set of signals to block. They are added to the current signal mask.</p> <p><code>SIG_UNBLOCK</code> <i>set</i> corresponds to a set of signals to unblock. These signals are deleted from the current signal mask.</p> <p><code>SIG_SETMASK</code> <i>set</i> corresponds to the new signal mask. The current signal mask is replaced by <i>set</i>.</p> <p>If the value of <i>oset</i> is not <code>NULL</code>, it points to the location where the previous signal mask is stored.</p> |
| RETURN VALUES | Upon successful completion, the thr_sigsetmask() function returns 0. Otherwise, it returns a non-zero value. |
| ERRORS | <p>The thr_sigsetmask() function will fail if:</p> <p>EINVAL The value of <i>how</i> is not defined and <i>oset</i> is <code>NULL</code>.</p> |
| EXAMPLES | <p>EXAMPLE 1 The following example shows how to create a default thread that can serve as a signal catcher/handler with its own signal mask. <i>new</i> will have a different value from the creator’s signal mask.</p> <p>As POSIX threads and Solaris threads are fully compatible even within the same process, this example uses pthread_create(3T) if you execute <code>a.out 0</code>, or thr_create(3T) if you execute <code>a.out 1</code>.</p> <p>In this example:</p> |

- **sigemptyset(3C)** initializes a null signal set, new. **sigaddset(3C)** packs the signal, **SIGINT**, into that new set.
- Either **pthread_sigmask()** or **thr_sigsetmask()** is used to mask the signal, **SIGINT** (CTRL-C), from the calling thread, which is **main()**. The signal is masked to guarantee that only the new thread will receive this signal.
- **pthread_create()** or **thr_create()** creates the signal-handling thread.
- Using **pthread_join(3T)** or **thr_join(3T)**, **main()** then waits for the termination of that signal-handling thread, whose ID number is **user_threadID**; after which, **main()** will **sleep(3C)** for 2 seconds, and then the program terminates.
- The signal-handling thread, handler:
 - Assigns the handler **interrupt()** to handle the signal **SIGINT**, by the call to **sigaction(2)**.
 - Resets its own signal set to *not block* the signal, **SIGINT**.
 - Sleeps for 8 seconds to allow time for the user to deliver the signal, **SIGINT**, by pressing the CTRL-C.

```

/* cc thisfile.c -lthread -lpthread */
#define _REENTRANT /* basic first 3-lines for threads */
#include <pthread.h>
#include <thread.h>

thread_t user_threadID;
sigset_t new;
void *handler(), interrupt();

main( int argc, char *argv[] ){
    test_argv(argv[1]);

    sigemptyset(&new);
    sigaddset(&new, SIGINT);
    switch(*argv[1]) {

        case '0': /* POSIX */
            pthread_sigmask(SIG_BLOCK, &new, NULL);
            pthread_create(&user_threadID, NULL, handler, argv[1]);
            pthread_join(user_threadID, NULL);
            break;

        case '1': /* Solaris */
            thr_sigsetmask(SIG_BLOCK, &new, NULL);
            thr_create(NULL, 0, handler, argv[1], 0, &user_threadID);
            thr_join(user_threadID, NULL, NULL);
            break;
    } /* switch */

    printf("thread handler, # %d, has exited\n",user_threadID);
    sleep(2);

```

```

        printf("main thread, # %d is done\n", thr_self());
    } /* end main */

    struct sigaction act;

    void *
    handler(char argv1[])
    {
        act.sa_handler = interrupt;
        sigaction(SIGINT, &act, NULL);
        switch(*argv1){
            case '0': /* POSIX */
                pthread_sigmask(SIG_UNBLOCK, &new, NULL);
                break;
            case '1': /* Solaris */
                thr_sigsetmask(SIG_UNBLOCK, &new, NULL);
                break;
        }
        printf("\n Press CTRL-C to deliver SIGINT signal to the process\n");
        sleep(8); /* give user time to hit CTRL-C */
    }

    void
    interrupt(int sig)
    {
        printf("thread %d caught signal %d\n", thr_self(), sig);
    }

    void test_argv(char argv1[] ) {
        if(argv1 == NULL) {
            printf("use 0 as arg1 to use thr_create();\n \
                or use 1 as arg1 to use pthread_create()\n");
            exit(NULL);
        }
    }
}

```

EXAMPLE 2

In the last example, the handler thread served as a signal-handler while also taking care of activity of its own (in this case, sleeping, although it could have been some other activity). A thread could be completely dedicated to signal-handling simply by waiting for the delivery of a selected signal by blocking with `sigwait(2)`. The two subroutines in the previous example, `handler()` and `interrupt()`, could have been replaced with the following routine:

```

void *
handler()
{ int signal;
  printf("thread %d waiting for you to press the CTRL-C keys\n", thr_self());
  sigwait(&new, &signal);
  printf("thread %d has received the signal %d \n", thr_self(), signal);
}
/*pthread_create() and thr_create() would use NULL instead of argv[1]
for the arg passed to handler() */

```

In this routine, one thread is dedicated to catching and handling the signal specified by the set `new`, which allows `main()` and all of its other sub-threads, created *after* `pthread_sigmask()` or `thr_sigsetmask()` masked that signal, to continue uninterrupted. Any use of `sigwait(2)` should be such that all threads block the signals passed to `sigwait(2)` at all times. Only the thread that calls `sigwait()` will get the signals. The call to `sigwait(2)` takes two arguments.

For this type of background dedicated signal-handling routine, you may wish to use a Solaris daemon thread by passing the argument `THR_DAEMON` to `thr_create()`.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------------|
| MT-Level | MT-Safe and Async-Signal-Safe |

SEE ALSO

`sigaction(2)`, `sigprocmask(2)`, `sigwait(2)`, `cond_wait(3T)`, `pthread_create(3T)`, `pthread_join(3T)`, `pthread_self(3T)`, `sigsetops(3C)`, `sleep(3C)`, `attributes(5)`, `standards(5)`

NOTES

It is not possible to block signals that cannot be ignored (see `sigaction(2)`). If using the threads library, it is not possible to block the signals `SIGLWP` or `SIGCANCEL`, which are reserved by the threads library. Additionally, it is impossible to unblock the signal `SIGWAITING`, which is always blocked on all threads. This restriction is quietly enforced by the threads library.

Using `sigwait(2)` in a dedicated thread allows asynchronously generated signals to be managed synchronously; however, `sigwait(2)` should never be used to manage synchronously generated signals.

Synchronously generated signals are exceptions that are generated by a thread and are directed at the thread causing the exception. Since `sigwait()` blocks waiting for signals, the blocking thread cannot receive a synchronously generated signal.

If `sigprocmask(2)` is used in a multi-threaded program, it will be the same as if `thr_sigsetmask()` or `pthread_sigmask()` has been called. POSIX leaves the semantics of the call to `sigprocmask(2)` unspecified in a multi-threaded process, so programs that care about POSIX portability should not depend on this semantic.

If a signal is delivered while a thread is waiting on a condition variable, the `cond_wait()` will be interrupted (see `cond_wait(3T)`) and the handler will be

executed. The handler should assume that the lock protecting the condition variable is held.

Signals which are generated synchronously should not be masked. If such a signal is blocked and delivered, the receiving process is killed.

A thread directed `SIGALRM` generated because of a realtime interval timer or process alarm clock is not maskable by a signal masking function, such as `thr_sigsetmask(3T)`, or `sigprocmask(2)`. See `alarm(2)` and `setitimer(2)`.

| NAME | thr_stksegment – get thread stack bottom and stack size | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc - mt [flag...] file...[library...] #include <thread.h> #include <sys/signal.h> int thr_stksegment(stack_t*);</pre> | | | | |
| DESCRIPTION | <p>The stack information provided by thr_stksegment() is typically used by debuggers, garbage collectors, and similar applications. Most applications should not require such information. The bottom of the thread stack returned by thr_stksegment() points to a part of the stack which may contain data maintained by <code>libthread</code>. The user's thread stack starts at a point below the bottom of the stack as returned by <code>thr_stksegment()</code>.</p> | | | | |
| RETURN VALUES | <p>The thr_stksegment() function returns 0 if both the thread stack bottom and stack size were successfully retrieved. Otherwise, it returns a non-zero error code.</p> | | | | |
| ERRORS | <p>The thr_stksegment() function will fail if:</p> <p>EAGAIN The stack information for the thread is not available because the thread's initialization is not yet complete, or the thread is an internal thread.</p> <p>The thr_stksegment() function may fail if:</p> <p>EFAULT A system call used to get the stack information failed because a bad address was passed to it.</p> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | thr_create(3T) , attributes(5) | | | | |

| | |
|----------------------|---|
| NAME | thr_suspend, thr_continue – suspend or continue thread execution |
| SYNOPSIS | <pre>cc - mt [flag ...] file ...[library ...] #include <thread.h> int thr_suspend(thread_t target_thread); int thr_continue(thread_t target_thread);</pre> |
| DESCRIPTION | <p>The thr_suspend() function immediately suspends the execution of the thread specified by <i>target_thread</i> . On successful return from thr_suspend() , the suspended thread is no longer executing. Once a thread is suspended, subsequent calls to thr_suspend() have no effect.</p> <p>The thr_continue() function resumes the execution of a suspended thread. Once a suspended thread is continued, subsequent calls to thr_continue() have no effect.</p> <p>A suspended thread will not be awakened by a signal. The signal stays pending until the execution of the thread is resumed by thr_continue() .</p> |
| RETURN VALUES | If successful, the thr_suspend() and thr_continue() functions return 0 . Otherwise, a non-zero value is returned to indicate the error. |
| ERRORS | <p>The thr_suspend() or thr_continue() functions will fail if:</p> <p>ESRCH <i>target_thread</i> cannot be found in the current process.</p> <p>ECANCELED <i>target_thread</i> was not suspended because a subsequent thr_continue() occurred before the suspend completed.</p> <p>EINVAL When thr_continue() returns EINVAL , <i>target_thread</i> has died and thr_join() must be called on it to reclaim its resources.</p> <p>The thr_suspend() function will fail if:</p> <p>EDEADLK Suspending <i>target_thread</i> will cause all threads in the process to be suspended.</p> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`thr_create(3T)`, `thr_join(3T)`, `attributes(5)`, `standards(5)`

NAME thr_yield – yield to another thread

SYNOPSIS cc - mt [*flag...*] *file...*[*library...*]

```
#include <thread.h>
```

```
void thr_yield(void);
```

DESCRIPTION The **thr_yield()** function causes the current thread to yield its execution in favor of another thread with the same or greater priority.

RETURN VALUES The **thr_yield()** function returns nothing and does not set **errno**.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **thr_setprio(3T)**, **attributes(5)**, **standards(5)**

| | |
|----------------------|---|
| NAME | tigetflag, tigetnum, tigetstr, tparm – return the value of a terminfo capability |
| SYNOPSIS | <pre>#include <term.h> int tigetflag(char * capname); int tigetnum(char * capname); char * tigetstr(char * capname); char * tparm(char * cap, long p1, long p2, long p3, long p4, long p5, long p6, long p7, long p8, long p9);</pre> |
| PARAMETERS | <p>capname Is the name of the <code>terminfo</code> capability for which the value is required.</p> <p>cap Is a pointer to a string capability.</p> <p>p1 ... p9 Are the parameters to be instantiated.</p> |
| DESCRIPTION | <p>The tigetflag() , tigetnum() , and tigetstr() functions return values for <code>terminfo</code> capabilities passed to them.</p> <p>The following null-terminated arrays contain the <i>capnames</i> , the <code>termcap</code> codes and full C names for each of the <code>terminfo</code> variables.</p> <pre>char *boolnames, *boolcodes, *boolfnames char *numnames, *numcodes, *numfnames char *strnames, *strcodes, *strfnames</pre> <p>The tparm() function instantiates a parameterized string using nine arguments. The string is suitable for output processing by tputs() .</p> |
| RETURN VALUES | <p>On success, the tigetflag() , tigetnum() , and tigetstr() functions return the specified <code>terminfo</code> capability.</p> <p>tigetflag() returns <code>-1</code> if <i>capname</i> is not a Boolean capability.</p> <p>tigetnum() returns <code>-2</code> if <i>capname</i> is not a numeric capability.</p> <p>tigetstr() returns <code>(char *) -1</code> if <i>capname</i> is not a string capability.</p> <p>On success, the tparm() function returns <i>cap</i> in a static buffer with the parameterization resolved. Otherwise, it returns a null pointer.</p> |

ERRORS | None.

SEE ALSO | `tgetent(3XC)` , `terminfo(4)`

| | | | | | | | |
|-----------------------------|---|-----------------------------|--|----------------------------|---|-------------------------|---------------------------------|
| NAME | timer_create - create a timer | | | | | | |
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <signal.h> #include <time.h> int timer_create(clockid_t clock_id, struct sigevent *evp, timer_t *timerid);</pre> | | | | | | |
| DESCRIPTION | <p>The timer_create() function creates a timer using the specified clock, <i>clock_id</i>, as the timing base. The timer_create() function returns, in the location referenced by <i>timerid</i>, a timer ID of type <code>timer_t</code> used to identify the timer in timer requests. This timer ID will be unique within the calling process until the timer is deleted. The particular clock, <i>clock_id</i>, is defined in <code><time.h></code>. The timer whose ID is returned will be in a disarmed state upon return from timer_create().</p> <p>The <i>evp</i> argument, if non-NULL, points to a <code>sigevent</code> structure. This structure, allocated by the application, defines the asynchronous notification that will occur when the timer expires. If the <i>evp</i> argument is NULL, the effect is as if the <i>evp</i> argument pointed to a <code>sigevent</code> structure with the <code>sigev_notify</code> member having the value <code>SIGEV_SIGNAL</code>, the <code>sigev_signo</code> having a default signal number, and the <code>sigev_value</code> member having the value of the timer ID, <i>timerid</i>.</p> <p>The system defines a set of clocks that can be used as timing bases for per-process timers. The following values for <i>clock_id</i> are supported:</p> <table border="0" style="margin-left: 20px;"> <tr> <td><code>CLOCK_REALTIME</code></td> <td>wall clock, not bound</td> </tr> <tr> <td><code>CLOCK_VIRTUAL</code></td> <td>user CPU usage clock</td> </tr> <tr> <td><code>CLOCK_PROF</code></td> <td>user and system CPU usage clock</td> </tr> </table> <p>Timers are not inherited by a child process across a <code>fork(2)</code> and are disarmed and deleted by a call to one of the <code>exec</code> functions (see <code>exec(2)</code>).</p> | <code>CLOCK_REALTIME</code> | wall clock, not bound | <code>CLOCK_VIRTUAL</code> | user CPU usage clock | <code>CLOCK_PROF</code> | user and system CPU usage clock |
| <code>CLOCK_REALTIME</code> | wall clock, not bound | | | | | | |
| <code>CLOCK_VIRTUAL</code> | user CPU usage clock | | | | | | |
| <code>CLOCK_PROF</code> | user and system CPU usage clock | | | | | | |
| RETURN VALUES | If the call succeeds, timer_create() returns 0 and updates the location referenced by <i>timerid</i> to a <code>timer_t</code> , which can be passed to the per-process timer calls. If an error occurs, the function returns -1 and sets <code>errno</code> to indicate the error. The value of <i>timerid</i> is undefined if an error occurs. | | | | | | |
| ERRORS | <p>The timer_create() function will fail if:</p> <table border="0" style="margin-left: 20px;"> <tr> <td>EAGAIN</td> <td>The system lacks sufficient signal queuing resources to honor the request, or the calling process has already created all of the timers it is allowed by the system.</td> </tr> <tr> <td>EINVAL</td> <td>The specified clock ID, <i>clock_id</i>, is not defined.</td> </tr> </table> | EAGAIN | The system lacks sufficient signal queuing resources to honor the request, or the calling process has already created all of the timers it is allowed by the system. | EINVAL | The specified clock ID, <i>clock_id</i> , is not defined. | | |
| EAGAIN | The system lacks sufficient signal queuing resources to honor the request, or the calling process has already created all of the timers it is allowed by the system. | | | | | | |
| EINVAL | The specified clock ID, <i>clock_id</i> , is not defined. | | | | | | |

ENOSYS The `timer_create()` function is not supported by the system.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |

SEE ALSO

`exec(2)`, `fork(2)`, `time(2)`, `clock_gettime(3R)`, `signal(3C)`,
`timer_delete(3R)`, `timer_settime(3R)`, `attributes(5)`

NOTES

Due to the way that signals are handled, if two timers expire at approximately the same time, the signal handler might not detect both of them.

| NAME | timer_delete – delete a timer | | | | |
|----------------------|--|----------------|-----------------|----------|-------------------------|
| SYNOPSIS | <pre>cc [flag...] file... -lrt [library...] #include <time.h> int timer_delete(timer_t timerid);</pre> | | | | |
| DESCRIPTION | The timer_delete() function deletes the specified timer, <i>timerid</i> , previously created by the timer_create(3R) function. If the timer is armed when timer_delete() is called, the behavior will be as if the timer is automatically disarmed before removal. The disposition of pending signals for the deleted timer is unspecified. | | | | |
| RETURN VALUES | If successful, the function returns 0. Otherwise, the function returns -1 and sets <code>errno</code> to indicate the error. | | | | |
| ERRORS | <p>The timer_delete() function will fail if:</p> <p>EINVAL The timer ID specified by <i>timerid</i> is not a valid timer ID.</p> <p>ENOSYS The timer_delete() function is not supported by the system.</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe with exceptions</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe with exceptions |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe with exceptions | | | | |
| SEE ALSO | timer_create(3R) , attributes(5) | | | | |

| | |
|--------------------|--|
| NAME | timer_settime, timer_gettime, timer_getoverrun – per-process timers |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lrt [<i>library</i> ...] #include <time.h> int timer_settime(timer_t <i>timerid</i>, int <i>flags</i>, const struct itimerspec * <i>value</i>, struct itimerspec * <i>ovalue</i>); int timer_gettime(timer_t <i>timerid</i>, struct itimerspec * <i>value</i>); int timer_getoverrun(timer_t <i>timerid</i>);</pre> |
| DESCRIPTION | <p>The timer_settime() function sets the time until the next expiration of the timer specified by <i>timerid</i> from the <i>it_value</i> member of the <i>value</i> argument and arm the timer if the <i>it_value</i> member of <i>value</i> is non-zero. If the specified timer was already armed when timer_settime() is called, this call resets the time until next expiration to the <i>value</i> specified. If the <i>it_value</i> member of <i>value</i> is 0, the timer is disarmed. The effect of disarming or resetting a timer on pending expiration notifications is unspecified.</p> <p>If the flag <code>TIMER_ABSTIME</code> is not set in the argument <i>flags</i>, timer_settime() behaves as if the time until next expiration is set to be equal to the interval specified by the <i>it_value</i> member of <i>value</i>. That is, the timer expires in <i>it_value</i> nanoseconds from when the call is made. If the flag <code>TIMER_ABSTIME</code> is set in the argument <i>flags</i>, timer_settime() behaves as if the time until next expiration is set to be equal to the difference between the absolute time specified by the <i>it_value</i> member of <i>value</i> and the current value of the clock associated with <i>timerid</i>. That is, the timer expires when the clock reaches the value specified by the <i>it_value</i> member of <i>value</i>. If the specified time has already passed, the function succeeds and the expiration notification is made.</p> <p>The reload value of the timer is set to the value specified by the <i>it_interval</i> member of <i>value</i>. When a timer is armed with a non-zero <i>it_interval</i>, a periodic (or repetitive) timer is specified.</p> <p>Time values that are between two consecutive non-negative integer multiples of the resolution of the specified timer will be rounded up to the larger</p> |

multiple of the resolution. Quantization error will not cause the timer to expire earlier than the rounded time value.

If the argument *ovalue* is not `NULL`, the function `timer_settime()` stores, in the location referenced by *ovalue*, a value representing the previous amount of time before the timer would have expired or 0 if the timer was disarmed, together with the previous timer reload value. The members of *ovalue* are subject to the resolution of the timer, and they are the same values that would be returned by a `timer_gettime()` call at that point in time.

The `timer_gettime()` function stores the amount of time until the specified timer, *timerid*, expires and the reload value of the timer into the space pointed to by the *value* argument. The `it_value` member of this structure contains the amount of time before the timer expires, or 0 if the timer is disarmed. This value is returned as the interval until timer expiration, even if the timer was armed with absolute time. The `it_interval` member of *value* contains the reload value last set by `timer_settime()`.

Only a single signal will be queued to the process or LWP for a given timer at any point in time. When a timer for which a signal is still pending expires, no signal will be queued, and a timer overrun occurs. When a timer expiration signal is delivered to or accepted by a process, the `timer_getoverrun()` function returns the timer expiration overrun count for the specified timer. The overrun count returned contains the number of extra timer expirations that occurred between the time the signal was generated (queued) and when it was delivered or accepted, up to but not including an implementation-dependent maximum of `DELAFTIMER_MAX`. If the number of such extra expirations is greater than or equal to `DELAFTIMER_MAX`, then the overrun count will be set to `DELAFTIMER_MAX`. The value returned by `timer_getoverrun()` applies to the most recent expiration signal delivery or acceptance for the timer. If no expiration signal has been delivered for the timer, the meaning of the overrun count returned is undefined.

RETURN VALUES

If the `timer_settime()` or `timer_gettime()` functions succeed, 0 is returned. If an error occurs for either of these functions, -1 is returned, and `errno` is set to indicate the error. If the `timer_getoverrun()` function succeeds, it returns the timer expiration overrun count as explained above.

ERRORS

The `timer_settime()`, `timer_gettime()` and `timer_getoverrun()` functions will fail if:

EINVAL The *timerid* argument does not correspond to a timer returned by `timer_create(3R)` but not yet deleted by `timer_delete(3R)`.

ENOSYS The `timer_settime()`, `timer_gettime()`, and `timer_getoverrun()` functions are not supported by the system. The `timer_settime()` function will fail if:

EINVAL A *value* structure specified a nanosecond value less than zero or greater than or equal to 1000 million.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------|
| MT-Level | Async-Signal-Safe |

SEE ALSO

`clock_settime(3R)`, `timer_create(3R)`, `timer_delete(3R)`,
`attributes(5)`, `time(5)`

| | |
|----------------------|--|
| NAME | times – get process times |
| SYNOPSIS | <pre> /usr/ucb/cc [flag ...] file ... #include <sys/param.h> #include <sys/types.h> #include <sys/times.h> int times(<i>tmsp</i>); register struct tms *<i>tmsp</i>; </pre> |
| DESCRIPTION | <p>times() returns time-accounting information for the current process and for the terminated child processes of the current process. All times are reported in clock ticks. The number of clock ticks per second is defined by the variable <code>CLK_TCK</code>, found in the header <code><limits.h></code>.</p> <p>A structure with the following members is returned by times():</p> <pre> time_t tms_utime; /* user time */ time_t tms_stime; /* system time */ time_t tms_cutime; /* user time, children */ time_t tms_cstime; /* system time, children */ </pre> <p>The children's times are the sum of the children's process times and their children's times.</p> |
| RETURN VALUES | <p>times() returns</p> <p>0 on success.</p> <p>-1 on failure.</p> |
| SEE ALSO | <code>time(1)</code> , <code>time(2)</code> , <code>wait(2)</code> , <code>getrusage(3C)</code> |
| NOTES | <p>Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-threaded applications is unsupported.</p> <p>times() has been superseded by <code>getrusage(3C)</code>.</p> |

| | |
|--------------------|---|
| NAME | t_listen – listen for a connection indication |
| SYNOPSIS | <pre>#include <xti.h> int t_listen(int fd, struct t_call *call);</pre> |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>This function listens for a connection indication from a calling transport user. The argument <i>fd</i> identifies the local transport endpoint where connection indications arrive, and on return, <i>call</i> contains information describing the connection indication. The parameter <i>call</i> points to a <code>t_call</code> structure which contains the following members:</p> <pre>struct netbuf addr; struct netbuf opt; struct netbuf udata; int sequence;</pre> <p>In <i>call</i>, <i>addr</i> returns the protocol address of the calling transport user. This address is in a format usable in future calls to <code>t_connect(3N)</code>. Note, however that <code>t_connect(3N)</code> may fail for other reasons, for example <code>TADDRBUSY</code>. <i>opt</i> returns options associated with the connection indication, <i>udata</i> returns any user data sent by the caller on the connection request, and <i>sequence</i> is a number that uniquely identifies the returned connection indication. The value of <i>sequence</i> enables the user to listen for multiple connection indications before responding to any of them.</p> <p>Since this function returns values for the <i>addr</i>, <i>opt</i> and <i>udata</i> fields of <i>call</i>, the <i>maxlen</i> field of each must be set before issuing the <code>t_listen()</code> to indicate the maximum size of the buffer for each. If the <i>maxlen</i> field of <i>call</i>→<i>addr</i>, <i>call</i>→<i>opt</i> or <i>call</i>→<i>udata</i> is set to zero, no information is returned for this parameter.</p> <p>By default, <code>t_listen()</code> executes in synchronous mode and waits for a connection indication to arrive before returning to the user. However, if <code>O_NONBLOCK</code> is set via <code>t_open(3N)</code> or <code>fcntl(2)</code>, <code>t_listen()</code> executes asynchronously, reducing to a poll for existing connection indications. If none are available, it returns <code>-1</code> and sets <code>t_errno</code> to <code>TNODATA</code>.</p> |

RETURN VALUES

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and `t_errno` is set to indicate an error.

VALID STATES

T_IDLE, T_INCON

ERRORS

On failure, `t_errno` is set to one of the following:

| | |
|-------------|---|
| TBADF | The specified file descriptor does not refer to a transport endpoint. |
| TBADQLEN | The argument <i>qlen</i> of the endpoint referenced by <i>fd</i> is zero. |
| TBUFOVFLW | The number of bytes allocated for an incoming argument (<i>maxlen</i>) is greater than 0 but not sufficient to store the value of that argument. The provider's state, as seen by the user, changes to T_INCON, and the connection indication information to be returned in <i>call</i> is discarded. The value of <i>sequence</i> returned can be used to do a <code>t_snddis(3N)</code> . |
| TLOOK | An asynchronous event has occurred on this transport endpoint and requires immediate attention. |
| TNODATA | O_NONBLOCK was set, but no connection indications had been queued. |
| TNOTSUPPORT | This function is not supported by the underlying transport provider. |
| TOUTSTATE | The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. |
| TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). |
| TQFULL | The maximum number of outstanding connection indications has been reached for the endpoint referenced by <i>fd</i> . Note that a subsequent call to <code>t_listen()</code> may block until another incoming connection indication is available. This can only occur if at least one of the outstanding connection indications becomes no longer outstanding, for example through a call to <code>t_accept(3N)</code> . |
| TSYSERR | A system error has occurred during execution of this function. |

**TLI
COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```
#include <tiuser.h>
```

**Error Description
Values**

The `t_errno` values `TPROTO0`, `TBADQLEN`, and `TQFULL` can be set by the XTI interface but not by the TLI interface.

A `t_errno` value that this routine can return under different circumstances than its XTI counterpart is `TBUFOVFLW`. It can be returned even when the `maxlen` field of the corresponding buffer has been set to zero.

Option Buffers

The format of the options in an `opt` buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not fix the buffer format.

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`fcntl(2)`, `t_accept(3N)`, `t_alloc(3N)`, `t_bind(3N)`, `t_connect(3N)`, `t_open(3N)`, `t_optmgmt(3N)`, `t_rcvconnect(3N)`, `t_snddis(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

WARNINGS

Some transport providers do not differentiate between a connection indication and the connection itself. If this is the case, a successful return of `t_listen()` indicates an existing connection.

| | | | | | | | | | | | | | | | | | | | |
|----------------------|--|----------|---------------------------------|-----------|--------------------------------|--------|-----------------------|----------|--------------------------|--------------|-------------------------|---------|----------------------------|----------|-----------------------------|----------|--|------------|--|
| NAME | t_look – look at the current event on a transport endpoint | | | | | | | | | | | | | | | | | | |
| SYNOPSIS | #include <xti.h> int t_look(int <i>fd</i>); | | | | | | | | | | | | | | | | | | |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>This function returns the current event on the transport endpoint specified by <i>fd</i>. This function enables a transport provider to notify a transport user of an asynchronous event when the user is calling functions in synchronous mode. Certain events require immediate notification of the user and are indicated by a specific error, TLOOK, on the current or next function to be executed.</p> <p>This function also enables a transport user to poll a transport endpoint periodically for asynchronous events.</p> | | | | | | | | | | | | | | | | | | |
| RETURN VALUES | <p>Upon success, t_look() returns a value that indicates which of the allowable events has occurred, or returns zero if no event exists. One of the following events is returned:</p> <table border="0"> <tr> <td>T_LISTEN</td> <td>Connection indication received.</td> </tr> <tr> <td>T_CONNECT</td> <td>Connect confirmation received.</td> </tr> <tr> <td>T_DATA</td> <td>Normal data received.</td> </tr> <tr> <td>T_EXDATA</td> <td>Expedited data received.</td> </tr> <tr> <td>T_DISCONNECT</td> <td>Disconnection received.</td> </tr> <tr> <td>T_UDERR</td> <td>Datagram error indication.</td> </tr> <tr> <td>T_ORDREL</td> <td>Orderly release indication.</td> </tr> <tr> <td>T_GODATA</td> <td>Flow control restrictions on normal data flow that led to a TFLOW error have been lifted. Normal data may be sent again.</td> </tr> <tr> <td>T_GOEXDATA</td> <td>Flow control restrictions on expedited data flow that led to a TFLOW error have been lifted. Expedited data may be sent again.</td> </tr> </table> <p>On failure, -1 is returned and <code>t_errno</code> is set to indicate the error.</p> | T_LISTEN | Connection indication received. | T_CONNECT | Connect confirmation received. | T_DATA | Normal data received. | T_EXDATA | Expedited data received. | T_DISCONNECT | Disconnection received. | T_UDERR | Datagram error indication. | T_ORDREL | Orderly release indication. | T_GODATA | Flow control restrictions on normal data flow that led to a TFLOW error have been lifted. Normal data may be sent again. | T_GOEXDATA | Flow control restrictions on expedited data flow that led to a TFLOW error have been lifted. Expedited data may be sent again. |
| T_LISTEN | Connection indication received. | | | | | | | | | | | | | | | | | | |
| T_CONNECT | Connect confirmation received. | | | | | | | | | | | | | | | | | | |
| T_DATA | Normal data received. | | | | | | | | | | | | | | | | | | |
| T_EXDATA | Expedited data received. | | | | | | | | | | | | | | | | | | |
| T_DISCONNECT | Disconnection received. | | | | | | | | | | | | | | | | | | |
| T_UDERR | Datagram error indication. | | | | | | | | | | | | | | | | | | |
| T_ORDREL | Orderly release indication. | | | | | | | | | | | | | | | | | | |
| T_GODATA | Flow control restrictions on normal data flow that led to a TFLOW error have been lifted. Normal data may be sent again. | | | | | | | | | | | | | | | | | | |
| T_GOEXDATA | Flow control restrictions on expedited data flow that led to a TFLOW error have been lifted. Expedited data may be sent again. | | | | | | | | | | | | | | | | | | |

VALID STATES

ALL - apart from T_UNINIT.

ERRORS

On failure, `t_errno` is set to one of the following:

| | |
|---------|---|
| TBADF | The specified file descriptor does not refer to a transport endpoint. |
| TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). |
| TSYSERR | A system error has occurred during execution of this function. |

TLI COMPATIBILITY

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```
#include <tiuser.h>
```

Return Values

The return values that are defined by the XTI interface and cannot be returned by the TLI interface are:

```
T_GODATA
T_GOEXDATA
```

Error Description Values

The `t_errno` value that can be set by the XTI interface and cannot be set by the TLI interface is:

```
TPROTO
```

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`t_open(3N)`, `t_snd(3N)`, `t_sndudata(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

| | |
|----------------------|---|
| NAME | tmpfile – create a temporary file |
| SYNOPSIS | #include <stdio.h> FILE *tmpfile(void); |
| DESCRIPTION | The tmpfile() function creates a temporary file and opens a corresponding stream. The file will automatically be deleted when all references to the file are closed. The file is opened as in fopen(3S) for update (<i>w+</i>). The largest value that can be represented correctly in an object of type <code>off_t</code> will be established as the offset maximum in the open file description. |
| RETURN VALUES | Upon successful completion, tmpfile() returns a pointer to the stream of the file that is created. Otherwise, it returns a null pointer and sets <code>errno</code> to indicate the error. |
| ERRORS | The tmpfile() function will fail if: EINTR A signal was caught during the execution of tmpfile() . EMFILE There are <code>OPEN_MAX</code> file descriptors currently open in the calling process. ENFILE The maximum allowable number of files is currently open in the system. ENOSPC The directory or file system which would contain the new file cannot be expanded. The tmpfile() function may fail if: EMFILE There are <code>FOPEN_MAX</code> streams currently open in the calling process. ENOMEM Insufficient storage space is available. |
| USAGE | The stream refers to a file which is unlinked. If the process is killed in the period between file creation and unlinking, a permanent file may be left behind. The tmpfile() function has a transitional interface for 64-bit file offsets. See 1f64(5) . |
| SEE ALSO | unlink(2) , fopen(3S) , tmpnam(3S) , 1f64(5) |

| | |
|--------------------|---|
| NAME | tmpnam, tmpnam_r, tmpnam – create a name for a temporary file |
| SYNOPSIS | <pre>#include <stdio.h> char * tmpnam(char * s); char * tmpnam_r(char * s); char * tmpnam(const char * dir, const char * pfx);</pre> |
| DESCRIPTION | These functions generate file names that can safely be used for a temporary file. |
| tmpnam() | <p>The tmpnam() function always generates a file name using the path prefix defined as <code>P_tmpdir</code> in the <code><stdio.h></code> header. On Solaris systems, the default value for <code>P_tmpdir</code> is <code>/var/tmp</code>. If <code>s</code> is <code>NULL</code>, tmpnam() leaves its result in an internal static area and returns a pointer to that area. The next call to tmpnam() will destroy the contents of the area. If <code>s</code> is not <code>NULL</code>, it is assumed to be the address of an array of at least <code>L_tmpnam</code> bytes, where <code>L_tmpnam</code> is a constant defined in <code><stdio.h></code>; tmpnam() places its result in that array and returns <code>s</code>.</p> |
| tmpnam_r() | <p>The tmpnam_r() function has the same functionality as tmpnam() except that if <code>s</code> is a null pointer, the function returns <code>NULL</code>. This interface is as proposed in the POSIX.4a Draft #6 document, and is subject to change to be compliant to the standard when it is accepted.</p> |
| tmpnam() | <p>The tmpnam() function allows the user to control the choice of a directory. The argument <code>dir</code> points to the name of the directory in which the file is to be created. If <code>dir</code> is <code>NULL</code> or points to a string that is not a name for an appropriate directory, the path prefix defined as <code>P_tmpdir</code> in the <code><stdio.h></code> header is used. If that directory is not accessible, <code>/tmp</code> is used. If, however, the <code>TMPDIR</code> environment variable is set in the user's environment, its value is used as the temporary-file directory.</p> <p>Many applications prefer that temporary files have certain initial character sequences in their names. The <code>pfx</code> argument may be <code>NULL</code> or point to a string of up to five characters to be used as the initial characters of the temporary-file name.</p> <p>The tmpnam() function uses <code>malloc(3C)</code> to allocate space for the constructed file name, and returns a pointer to this area. Any pointer value returned from tmpnam() may serve as an argument to <code>free(3C)</code> (see <code>malloc(3C)</code>). If tmpnam() cannot return the expected result for any reason (for example, <code>malloc(3C)</code> failed), or if none of the above-mentioned attempts to find an appropriate directory was successful, a null pointer is returned. This function fails if there is not enough space.</p> |

USAGE | These functions generate a different file name each time they are called.

Files created using these functions and either `fopen(3S)` or `creat(2)` are temporary only in the sense that they reside in a directory intended for temporary use, and their names are unique. It is the user's responsibility to remove the file when its use is ended.

If called more than `TMP_MAX` (defined in `<stdio.h>`) times in a single process, these functions start recycling previously used names.

Between the time a file name is created and the file is opened, it is possible for some other process to create a file with the same name. This can never happen if that other process is using these functions or `mktemp(3C)` and the file names are chosen to render duplication by other means unlikely.

ATTRIBUTES | See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|------------------|
| MT-Level | See NOTES below. |

SEE ALSO | `creat(2)`, `unlink(2)`, `fopen(3S)`, `free(3C)`, `malloc(3C)`, `mktemp(3C)`, `tmpfile(3S)`, `attributes(5)`

NOTES | The `tmpnam()` function is unsafe in multithreaded applications. The `tempnam()` function is safe in multithreaded applications and should be used instead.

When compiling multithreaded applications, the `_REENTRANT` flag must be defined on the compile line. This flag should be used only with multithreaded applications.

| | |
|--------------------|---|
| NAME | tnfctl_buffer_alloc, tnfctl_buffer_dealloc – allocate or deallocate a buffer for trace data |
| SYNOPSIS | <pre>cc [flag ...] file ... -ltnfctl [library ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_buffer_alloc(tnfctl_handle_t * hndl, const char * trace_file_name, size_t trace_buffer_size); tnfctl_buffer_dealloc(tnfctl_handle_t * hndl);</pre> |
| DESCRIPTION | <p>tnfctl_buffer_alloc() allocates a buffer to which trace events are logged. When tracing a process using a tnfctl handle returned by tnfctl_pid_open(3X), tnfctl_exec_open(3X), tnfctl_indirect_open(3X), and tnfctl_internal_open(3X), <i>trace_file_name</i> is the name of the trace file to which trace events should be logged. It can be an absolute path specification or a relative path specification. If it is relative, the current working directory of the process that is calling tnfctl_buffer_alloc() is prefixed to <i>trace_file_name</i>. If the named trace file already exists, it is overwritten. For kernel tracing, that is, for a tnfctl handle returned by tnfctl_kernel_open(3X), trace events are logged to a trace buffer in memory; therefore, <i>trace_file_name</i> is ignored. Use tnfextract(1) to extract a kernel buffer into a file.</p> <p><i>trace_buffer_size</i> is the size in bytes of the trace buffer that should be allocated. An error is returned if an attempt is made to allocate a buffer when one already exists. tnfctl_buffer_alloc() affects the trace attributes; use tnfctl_trace_attrs_get(3X) to get the latest trace attributes after a buffer is allocated.</p> <p>tnfctl_buffer_dealloc() is used to deallocate a kernel trace buffer that is no longer needed. <i>hndl</i> must be a kernel handle, returned by tnfctl_kernel_open(3X). A process's trace file cannot be deallocated using tnfctl_buffer_dealloc(). Instead, once the trace file is no longer needed for analysis and after the process being traced exits, use rm(1) to remove the trace file. Do not remove the trace file while the process being traced is still alive. tnfctl_buffer_dealloc() affects the trace attributes; use tnfctl_trace_attrs_get(3X) to get the latest trace attributes after a buffer is deallocated.</p> |

| | |
|----------------------|---|
| | For a complete discussion of tnf tracing, see tracing(3X) . |
| RETURN VALUES | tnfctl_buffer_alloc() and tnfctl_buffer_dealloc() return <code>TNFCTL_ERR_NONE</code> upon success. |
| ERRORS | The following error codes apply to tnfctl_buffer_alloc() : |
| | <code>TNFCTL_ERR_BUFEXISTS</code> A buffer already exists. |
| | <code>TNFCTL_ERR_ACCES</code> Permission denied; could not create a trace file. |
| | <code>TNFCTL_ERR_SIZETOOSMALL</code> The <i>trace_buffer_size</i> requested is smaller than the minimum trace buffer size needed. Use <code>trace_min_size</code> of trace attributes in tnfctl_trace_attrs_get(3X) to determine the minimum size of the buffer. |
| | <code>TNFCTL_ERR_SIZETOOBIG</code> The requested trace file size is too big. |
| | <code>TNFCTL_ERR_BADARG</code> <i>trace_file_name</i> is NULL or the absolute path name is longer than <code>MAX PATH LEN</code> . |
| | <code>TNFCTL_ERR_ALLOCFAIL</code> A memory allocation failure occurred. |
| | <code>TNFCTL_ERR_INTERNAL</code> An internal error occurred. |
| | The following error codes apply to tnfctl_buffer_dealloc() : |
| | <code>TNFCTL_ERR_BADARG</code> <i>hndl</i> is not a kernel handle. |
| | <code>TNFCTL_ERR_NOBUF</code> No buffer exists to deallocate. |
| | <code>TNFCTL_ERR_BADDEALLOC</code> Cannot deallocate a trace buffer unless tracing is stopped. Use tnfctl_trace_state_set(3X) to stop tracing. |
| | <code>TNFCTL_ERR_INTERNAL</code> An internal error occurred. |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| Availability | SUNWtnfc |
| MT Level | MT-Safe |

SEE ALSO

`prex(1)`, `rm(1)`, `tnfextract(1)`, `TNF_PROBE(3X)`, `libtnfctl(3X)`,
`tnfctl_exec_open(3X)`, `tnfctl_indirect_open(3X)`,
`tnfctl_internal_open(3X)`, `tnfctl_kernel_open(3X)`,
`tnfctl_pid_open(3X)`, `tnfctl_trace_attrs_get(3X)`, `tracing(3X)`,
`attributes(5)`

| | | | | | | | | | |
|----------------------|--|---------------------|---|---------------------|-----------------------------|--------------------|--|---------------------|--|
| NAME | tnfctl_close – close a tnfctl handle | | | | | | | | |
| SYNOPSIS | <pre>cc [flag ...] file ... -ltnfctl [library ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_close(tnfctl_handle_t *hndl, tnfctl_targ_op_t action);</pre> | | | | | | | | |
| DESCRIPTION | <p>tnfctl_close() is used to close a tnfctl handle and to free up the memory associated with the handle. When the handle is closed, the tracing state and the states of the probes are not changed. tnfctl_close() can be used to close handles in any mode, that is, whether they were created by tnfctl_internal_open(3X), tnfctl_pid_open(3X), tnfctl_exec_open(3X), tnfctl_indirect_open(3X), or tnfctl_kernel_open(3X).</p> <p>The <i>action</i> argument is only used in direct mode, that is, if <i>hndl</i> was created by tnfctl_exec_open(3X) or tnfctl_pid_open(3X). In direct mode, <i>action</i> specifies whether the process will proceed, be killed, or remain suspended. <i>action</i> may have the following values:</p> <table border="0"> <tr> <td style="vertical-align: top;">TNFCTL_TARG_DEFAULT</td> <td>Kills the target process if <i>hndl</i> was created with tnfctl_exec_open(3X), but lets it continue if it was created with tnfctl_pid_open(3X).</td> </tr> <tr> <td style="vertical-align: top;">TNFCTL_TARG_KILL</td> <td>Kills the target process.</td> </tr> <tr> <td style="vertical-align: top;">TNFCTL_TARG_RESUME</td> <td>Allows the target process to continue.</td> </tr> <tr> <td style="vertical-align: top;">TNFCTL_TARG_SUSPEND</td> <td>Leaves the target process suspended. This is not a job control suspend. It is possible to attach to the process again with a debugger or with the tnfctl_pid_open(3X) interface. The target process can also be continued with prun(1).</td> </tr> </table> | TNFCTL_TARG_DEFAULT | Kills the target process if <i>hndl</i> was created with tnfctl_exec_open(3X) , but lets it continue if it was created with tnfctl_pid_open(3X) . | TNFCTL_TARG_KILL | Kills the target process. | TNFCTL_TARG_RESUME | Allows the target process to continue. | TNFCTL_TARG_SUSPEND | Leaves the target process suspended. This is not a job control suspend. It is possible to attach to the process again with a debugger or with the tnfctl_pid_open(3X) interface. The target process can also be continued with prun(1) . |
| TNFCTL_TARG_DEFAULT | Kills the target process if <i>hndl</i> was created with tnfctl_exec_open(3X) , but lets it continue if it was created with tnfctl_pid_open(3X) . | | | | | | | | |
| TNFCTL_TARG_KILL | Kills the target process. | | | | | | | | |
| TNFCTL_TARG_RESUME | Allows the target process to continue. | | | | | | | | |
| TNFCTL_TARG_SUSPEND | Leaves the target process suspended. This is not a job control suspend. It is possible to attach to the process again with a debugger or with the tnfctl_pid_open(3X) interface. The target process can also be continued with prun(1) . | | | | | | | | |
| RETURN VALUES | tnfctl_close() returns TNFCTL_ERR_NONE upon success. | | | | | | | | |
| ERRORS | <p>The following error codes apply to tnfctl_close():</p> <table border="0"> <tr> <td style="vertical-align: top;">TNFCTL_ERR_BADARG</td> <td>A bad argument was sent in <i>action</i>.</td> </tr> <tr> <td style="vertical-align: top;">TNFCTL_ERR_INTERNAL</td> <td>An internal error occurred.</td> </tr> </table> | TNFCTL_ERR_BADARG | A bad argument was sent in <i>action</i> . | TNFCTL_ERR_INTERNAL | An internal error occurred. | | | | |
| TNFCTL_ERR_BADARG | A bad argument was sent in <i>action</i> . | | | | | | | | |
| TNFCTL_ERR_INTERNAL | An internal error occurred. | | | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | | | | | |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| Availability | SUNWtnfc |
| MT Level | MT-Safe |

SEE ALSO

`prex(1)`, `prun(1)`, `TNF_PROBE(3X)`, `libtnfctl(3X)`,
`tnfctl_exec_open(3X)`, `tnfctl_indirect_open(3X)`,
`tnfctl_kernel_open(3X)`, `tnfctl_pid_open(3X)`, `tracing(3X)`,
`attributes(5)`

Programming Utilities Guide

| | |
|--------------------|---|
| NAME | tnfctl_indirect_open, tnfctl_check_libs – control probes of another process where caller provides /proc functionality |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -ltnfctl [<i>library</i> ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_indirect_open(void * <i>prochandle</i>, tnfctl_ind_config_t * <i>config</i>, tnfctl_handle_t ** <i>ret_val</i>); tnfctl_errcode_t tnfctl_check_libs(tnfctl_handle_t * <i>hndl</i>);</pre> |
| DESCRIPTION | <p>The interfaces tnfctl_indirect_open() and tnfctl_check_libs() are used to control probes in another process where the libtnfctl(3X) client has already opened proc(4) on the target process. An example of this is when the client is a debugger. Since these clients already use /proc on the target, libtnfctl(3X) cannot use /proc directly. Therefore, these clients must provide callback functions that can be used to inspect and to update the target process. The target process must load libtnfprobe.so.1 (defined in <tnf/tnfctl.h> as macro TNFCTL_LIBTNFPROBE).</p> <p>The first argument <i>prochandle</i> is a pointer to an opaque structure that is used in the callback functions that inspect and update the target process. This structure should encapsulate the state that the caller needs to use /proc on the target process (the /proc file descriptor). The second argument, <i>config</i>, is a pointer to</p> <pre>typedef struct tnfctl_ind_config { \011int (*p_read)(void *prochandle, paddr_t addr, char *buf, \011\011\011\011\011size_t size); \011int (*p_write)(void *prochandle, paddr_t addr, char *buf, \011\011\011\011\011size_t size); \011pid_t (*p_getpid)(void *prochandle); \011int (*p_obj_iter)(void *prochandle, tnfctl_ind_obj_f *func, \011\011\011\011\011void *client_data); } tnfctl_ind_config_t;</pre> <p>The first field <i>p_read</i> is the address of a function that can read <i>size</i> bytes at address <i>addr</i> in the target image into the buffer <i>buf</i>. The function should return 0 upon success.. The second field <i>p_write</i> is the address of a function that can</p> |

write `size` bytes at address `addr` in the target image from the buffer `buf`. The function should return 0 upon success. The third field `p_getpid` is the address of a function that should return the process id of the target process (`prochandle`). The fourth field `p_obj_iter` is the address of a function that iterates over all load objects and the executable by calling the callback function `func` with `client_data`. If `func` returns 0, `p_obj_iter` should continue processing link objects. If `func` returns any other value, `p_obj_iter` should stop calling the callback function and return that value. `p_obj_iter` should return 0 if it iterates over all load objects.

If a failure is returned by any of the functions in `config`, the error is propagated back as `PREX_ERR_INTERNAL` by the `libtnfctl` interface that called it.

The definition of `tnfctl_ind_obj_f` is:

```
typedef int
tnfctl_ind_obj_f(void *prochandle,
\011const struct tnfctl_ind_obj_info *obj
\011void *client_data);
typedef struct tnfctl_ind_obj_info {
\011int      objfd;\011\011\011/* -1 indicates fd not available */
\011paddr_t text_base;\011\011/* virtual addr of text segment */
\011paddr_t data_base;\011\011/* virtual addr of data segment */
\011const char *objname;    \011/* null-term. pathname to loadobj */
} tnfctl_ind_obj_info_t;
```

`objfd` should be the file descriptor of the load object or executable. If it is `-1`, then `objname` should be an absolute pathname to the load object or executable. If `objfd` is not closed by `libtnfctl`, it should be closed by the load object iterator function. `text_base` and `data_base` are the addresses where the text and data segments of the load object are mapped in the target process.

Whenever the target process opens or closes a dynamic object, the set of available probes may change. See `dlopen(3X)` and `dlclose(3X)`. In indirect mode, call `tnfctl_check_libs()` when such events occur to make `libtnfctl` aware of any changes. In other modes this is unnecessary but harmless. It is also harmless to call `tnfctl_check_libs()` when no such events have occurred.

RETURN VALUES

`tnfctl_indirect_open()` and `tnfctl_check_libs()` return `TNFCTL_ERR_NONE` upon success.

ERRORS

The following error codes apply to `tnfctl_indirect_open()`:

| | |
|---------------------------------------|--|
| <code>TNFCTL_ERR_ALLOCFAIL</code> | A memory allocation failure occurred. |
| <code>TNFCTL_ERR_BUSY</code> | Internal tracing is being used. |
| <code>TNFCTL_ERR_NOLIBTNFPROBE</code> | <code>libtnfprobe.so.1</code> is not loaded in the target process. |

TNFCTL_ERR_INTERNAL An internal error occurred.
 The following error codes apply to **tnfctl_check_libs()** :
 TNFCTL_ERR_ALLOCFAIL A memory allocation failure occurred.
 TNFCTL_ERR_INTERNAL An internal error occurred.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| Availability | SUNWtnfc |
| MT Level | MT-Safe |

SEE ALSO

prex(1), **TNF_PROBE(3X)**, **dlclose(3X)**, **dlopen(3X)**, **libtnfctl(3X)**,
tnfctl_probe_enable(3X), **tnfctl_probe_trace(3X)**, **tracing(3X)**,
proc(4), **attributes(5)**

Programming Utilities Guide Linker and Libraries Guide

NOTES

tnfctl_indirect_open() should only be called after the dynamic linker has mapped in all the libraries (rtld sync point) and called only after the process is stopped. Indirect process probe control assumes the target process is stopped whenever any **libtnfctl** interface is used on it. For example, when used for indirect process probe control, **tnfctl_probe_enable(3X)** and **tnfctl_probe_trace(3X)** should be called only for a process that is stopped.

| NAME | tnfctl_internal_open – create handle for internal process probe control | | | | | | | | |
|---------------------------------------|---|-----------------------------------|---------------------------------------|------------------------------|--|---------------------------------------|--|----------------------------------|-----------------------------|
| SYNOPSIS | <pre>cc [flag ...] file ... -ltnfctl [library ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_internal_open(tnfctl_handle_t **ret_val);</pre> | | | | | | | | |
| DESCRIPTION | <p>tnfctl_internal_open() returns in <i>ret_val</i> a pointer to an opaque handle that can be used to control probes in the same process as the caller (internal process probe control). The process must have <code>libtnfprobe.so.1</code> loaded. Probes in libraries that are brought in by <code>dlopen(3X)</code> will be visible after the library has been opened. Probes in libraries closed by a <code>dldclose(3X)</code> will not be visible after the library has been disassociated. See the NOTES section for more details.</p> | | | | | | | | |
| RETURN VALUES | tnfctl_internal_open() returns <code>TNFCTL_ERR_NONE</code> upon success. | | | | | | | | |
| ERRORS | <table border="0"> <tr> <td style="vertical-align: top;"><code>TNFCTL_ERR_ALLOCFAIL</code></td> <td>A memory allocation failure occurred.</td> </tr> <tr> <td style="vertical-align: top;"><code>TNFCTL_ERR_BUSY</code></td> <td>Another client is already tracing this program (internally or externally).</td> </tr> <tr> <td style="vertical-align: top;"><code>TNFCTL_ERR_NOLIBTNFPROBE</code></td> <td><code>libtnfprobe.so.1</code> is not linked in the target process.</td> </tr> <tr> <td style="vertical-align: top;"><code>TNFCTL_ERR_INTERNAL</code></td> <td>An internal error occurred.</td> </tr> </table> | <code>TNFCTL_ERR_ALLOCFAIL</code> | A memory allocation failure occurred. | <code>TNFCTL_ERR_BUSY</code> | Another client is already tracing this program (internally or externally). | <code>TNFCTL_ERR_NOLIBTNFPROBE</code> | <code>libtnfprobe.so.1</code> is not linked in the target process. | <code>TNFCTL_ERR_INTERNAL</code> | An internal error occurred. |
| <code>TNFCTL_ERR_ALLOCFAIL</code> | A memory allocation failure occurred. | | | | | | | | |
| <code>TNFCTL_ERR_BUSY</code> | Another client is already tracing this program (internally or externally). | | | | | | | | |
| <code>TNFCTL_ERR_NOLIBTNFPROBE</code> | <code>libtnfprobe.so.1</code> is not linked in the target process. | | | | | | | | |
| <code>TNFCTL_ERR_INTERNAL</code> | An internal error occurred. | | | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>Availability</td> <td>SUNWtnfc</td> </tr> <tr> <td>MT Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | Availability | SUNWtnfc | MT Level | MT-Safe | | |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | | | |
| Availability | SUNWtnfc | | | | | | | | |
| MT Level | MT-Safe | | | | | | | | |
| SEE ALSO | <p><code>ld(1)</code>, <code>prex(1)</code>, <code>TNF_PROBE(3X)</code>, <code>dlopen(3X)</code>, <code>dldclose(3X)</code>, <code>libtnfctl(3X)</code>, <code>tracing(3X)</code>, <code>attributes(5)</code></p> <p><i>Programming Utilities Guide Linker and Libraries Guide</i></p> | | | | | | | | |
| NOTES | <p><code>libtnfctl</code> interposes on <code>dlopen(3X)</code> and <code>dldclose(3X)</code> in order to be notified of libraries being dynamically opened and closed. This interposition is necessary for internal process probe control to update its list of probes. In these interposition functions, a lock is acquired to synchronize on traversal of the library list maintained by the runtime linker. To avoid deadlocking on this</p> | | | | | | | | |

lock, **tnfctl_internal_open()** should not be called from within the init section of a library that can be opened by **dlopen(3X)**.

Since interposition does not work as expected when a library is opened dynamically, **tnfctl_internal_open()** should not be used if the client opened `libtnfctl` through **dlopen(3X)**. In this case, the client program should be built with a static dependency on `libtnfctl`. Also, if the client program is explicitly linking in `-ldl`, it should link `-ltnfctl` before `-ldl`.

Probes in filtered libraries (see `ld(1)`) will not be seen because the filtee (backing library) is loaded lazily on the first symbol reference and not at process startup or **dlopen(3X)** time. A workaround is to call **tnfctl_check_libs(3X)** once the caller is sure that the filtee has been loaded.

| NAME | tnfctl_kernel_open – create handle for kernel probe control | | | | | | | | | | |
|--------------------------------------|---|-------------------------------|--|------------------------------|---|-----------------------------------|---------------------------|--------------------------------------|------------------------|----------------------------------|------------------------------|
| SYNOPSIS | <pre>cc [flag ...] file ... -ltnfctl [library ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_kernel_open(tnfctl_handle_t **ret_val);</pre> | | | | | | | | | | |
| DESCRIPTION | <p>tnfctl_kernel_open() starts a kernel tracing session and returns in <code>ret_val</code> an opaque handle that can be used to control tracing and probes in the kernel. Only one kernel tracing session is possible at a time on a given machine. An error code of <code>TNFCTL_ERR_BUSY</code> is returned if there is another process using kernel tracing. Use the command</p> <pre>fuser -f /dev/tnfctl</pre> <p>to print the process id of the process currently using kernel tracing. Only a superuser may use tnfctl_kernel_open(). An error code of <code>TNFCTL_ERR_ACCES</code> is returned if the caller does not have the necessary privileges.</p> | | | | | | | | | | |
| RETURN VALUES | tnfctl_kernel_open returns <code>TNFCTL_ERR_NONE</code> upon success. | | | | | | | | | | |
| ERRORS | <table border="0"> <tr> <td style="vertical-align: top;"><code>TNFCTL_ERR_ACCES</code></td> <td>Permission denied. Superuser privileges are needed for kernel tracing.</td> </tr> <tr> <td style="vertical-align: top;"><code>TNFCTL_ERR_BUSY</code></td> <td>Another client is currently using kernel tracing.</td> </tr> <tr> <td style="vertical-align: top;"><code>TNFCTL_ERR_ALLOCFAIL</code></td> <td>Memory allocation failed.</td> </tr> <tr> <td style="vertical-align: top;"><code>TNFCTL_ERR_FILENOTFOUND</code></td> <td>/dev/tnfctl not found.</td> </tr> <tr> <td style="vertical-align: top;"><code>TNFCTL_ERR_INTERNAL</code></td> <td>Some other failure occurred.</td> </tr> </table> | <code>TNFCTL_ERR_ACCES</code> | Permission denied. Superuser privileges are needed for kernel tracing. | <code>TNFCTL_ERR_BUSY</code> | Another client is currently using kernel tracing. | <code>TNFCTL_ERR_ALLOCFAIL</code> | Memory allocation failed. | <code>TNFCTL_ERR_FILENOTFOUND</code> | /dev/tnfctl not found. | <code>TNFCTL_ERR_INTERNAL</code> | Some other failure occurred. |
| <code>TNFCTL_ERR_ACCES</code> | Permission denied. Superuser privileges are needed for kernel tracing. | | | | | | | | | | |
| <code>TNFCTL_ERR_BUSY</code> | Another client is currently using kernel tracing. | | | | | | | | | | |
| <code>TNFCTL_ERR_ALLOCFAIL</code> | Memory allocation failed. | | | | | | | | | | |
| <code>TNFCTL_ERR_FILENOTFOUND</code> | /dev/tnfctl not found. | | | | | | | | | | |
| <code>TNFCTL_ERR_INTERNAL</code> | Some other failure occurred. | | | | | | | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>Availability</td> <td>SUNWtnfc</td> </tr> <tr> <td>MT Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | Availability | SUNWtnfc | MT Level | MT-Safe | | | | |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | | | | | |
| Availability | SUNWtnfc | | | | | | | | | | |
| MT Level | MT-Safe | | | | | | | | | | |
| SEE ALSO | prex(1) , fuser(1M) , TNF_PROBE(3X) , libtnfctl(3X) , tracing(3X) , tnf_kernel_probes (4) , attributes(5) | | | | | | | | | | |

| | |
|--------------------|---|
| NAME | tnfctl_pid_open, tnfctl_exec_open, tnfctl_continue – interfaces for direct probe and process control for another process |
| SYNOPSIS | <pre>cc [flag ...] file ... -ltnfctl [library ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_pid_open(pid_t pid, tnfctl_handle_t ** ret_val); tnfctl_errcode_t tnfctl_exec_open(const char * pgm_name, char * const * argv, char * const * envp, const char * libnfpobe_path, const char * ld_preload, tnfctl_handle_t ** ret_val); tnfctl_errcode_t tnfctl_continue(tnfctl_handle_t * hndl, tnfctl_event_t * evt, tnfctl_handle_t ** child_hndl);</pre> |
| DESCRIPTION | <p>tnfctl_pid_open() , tnfctl_exec_open() , and tnfctl_continue() are the interfaces used to create handles to control probes in another process (direct process probe control). Either tnfctl_pid_open() or tnfctl_exec_open() will return a handle in <i>ret_val</i> that can be used for probe control. On return of these calls, the process is stopped. tnfctl_continue() allows the process specified by <i>hndl</i> to continue execution.</p> <p>tnfctl_pid_open() attaches to a running process with process id of <i>pid</i> . The process is stopped on return of this call. tnfctl_pid_open() returns an error message if <i>pid</i> is the same as the calling process. See tnfctl_internal_open(3X) for information on internal process probe control. A pointer to an opaque handle is returned in <i>ret_val</i> , which can be used to control the process and the probes in the process. The target process must have <code>libtnfprobe.so.1</code> (defined in <code><tnf/tnfctl.h></code> as macro <code>TNFCTL_LIBTNFPROBE</code>) linked in for probe control to work.</p> <p>tnfctl_exec_open() is used to <code>exec(2)</code> a program and obtain a probe control handle. For probe control to work, the process image to be <code>exec</code> 'd must load <code>libtnfprobe.so.1</code> . The interface tnfctl_exec_open() makes it simple for the library to be loaded at process start up time. <i>pgm_name</i> is the command to <code>exec</code> . If <i>pgm_name</i> is not an absolute path, then the <code>\$ PATH</code> environment variable is used to find the <i>pgm_name</i> . <i>argv</i> is a null-terminated argument pointer, that is, it is a null-terminated array of pointers to null-terminated strings. These strings constitute the argument list available to the new process</p> |

image. *argv* must have at least one member, and it should point to a string that is the same as *pgm_name*. See `execve(2)`. *libnfp_path* is an optional argument, and if set, it should be the path to the directory that contains `libnfp.so.1`. There is no need for a trailing "/" in this argument. This argument is useful if `libnfp.so.1` is not installed in `/usr/lib`. *ld_preload* is a space-separated list of libraries to preload into the target program. This string should follow the syntax guidelines of the `LD_PRELOAD` environment variable. See `ld.so.1(1)`. The following illustrates how strings are concatenated to form the `LD_PRELOAD` environment variable in the new process image:

```
\011<current value of $LD_PRELOAD> + <space> +
\011
libnfp_path
+ "/libnfp.so.1" +
<space> +
\011
ld_preload
```

This option is useful for preloading interposition libraries that have probes in them.

envp is an optional argument, and if set, it is used for the environment of the target program. It is a null-terminated array of pointers to null-terminated strings. These strings constitute the environment of the new process image. See `execve(2)`. If *envp* is set, it overrides *ld_preload*. In this case, it is the caller's responsibility to ensure that `libnfp.so.1` is loaded into the target program. If *envp* is not set, the new process image inherits the environment of the calling process, except for `LD_PRELOAD`.

ret_val is the return argument which is the handle that can be used to control the process and the probes within the process. Upon return, the process is stopped before any user code, including `.init` sections, has been executed.

`tnfctl_continue()` is a blocking call and lets the target process referenced by *hdl* continue running. It can only be used on handles returned by `tnfctl_pid_open()` and `tnfctl_exec_open()` (direct process probe control). It returns when the target stops; the reason that the process stopped is returned in *evt*. This call is interruptible by signals. If it is interrupted, the process is stopped, and `TNFCTL_EVENT_EINTR` is returned in *evt*. The client of this library will have to decide which signal implies a stop to the target and catch that signal. Since a signal interrupts `tnfctl_continue()`, it will return, and the caller can decide whether or not to call `tnfctl_continue()` again.

`tnfctl_continue()` returns with an event of `TNFCTL_EVENT_DLOPEN`, `TNFCTL_EVENT_DLCLOSE`, `TNFCTL_EVENT_EXEC`, `TNFCTL_EVENT_FORK`, `TNFCTL_EVENT_EXIT`, or `TNFCTL_EVENT_TARGGONE`, respectively, when

the target program does a `dlopen(3X)`, `dlopen(3X)`, any flavor of `exec(2)`, `fork(2)` (or `fork1(2)`), `exit(2)`, or terminates unexpectedly. If the target program did an `exec(2)`, then the client needs to call `tnfctl_close(3X)` on the current handle leaving the target resumed, suspended, or killed (second argument to `tnfctl_close(3X)`). No other `libtnfctl` interface call can be used on the existing handle. If the client wants to control the `exec`'ed image, it should leave the old handle suspended, and use `tnfctl_pid_open()` to reattach to the same process. This new handle can then be used to control the `exec`'ed image. See `EXAMPLES` below for sample code. If the target process did a `fork(2)` or `fork1(2)`, and if control of the child process is not needed, then `child_hdl` should be `NULL`. If control of the child process is needed, then `child_hdl` should be set. If it is set, a pointer to a handle that can be used to control the child process is returned in `child_hdl`. The child process is stopped at the end of the `fork()` system call. See `EXAMPLES` for an example of this event.

RETURN VALUES

`tnfctl_pid_open()`, `tnfctl_exec_open()`, and `tnfctl_continue()` return `TNFCTL_ERR_NONE` upon success.

ERRORS

The following error codes apply to `tnfctl_pid_open()`:

| | |
|---------------------------------------|---|
| <code>TNFCTL_ERR_BADARG</code> | The <i>pid</i> specified is the same process. Use <code>tnfctl_internal_open(3X)</code> instead. |
| <code>TNFCTL_ERR_ACCES</code> | Permission denied. No privilege to connect to a setuid process. |
| <code>TNFCTL_ERR_ALLOCFAIL</code> | A memory allocation failure occurred. |
| <code>TNFCTL_ERR_BUSY</code> | Another client is already using <code>/proc</code> to control this process or internal tracing is being used. |
| <code>TNFCTL_ERR_NOTDYNAMIC</code> | The process is not a dynamic executable. |
| <code>TNFCTL_ERR_NOPROCESS</code> | No such target process exists. |
| <code>TNFCTL_ERR_NOLIBTNFPROBE</code> | <code>libtnfprobe.so.1</code> is not linked in the target process. |
| <code>TNFCTL_ERR_INTERNAL</code> | An internal error occurred. |

The following error codes apply to `tnfctl_exec_open()`:

| | |
|-----------------------------------|---------------------------------------|
| <code>TNFCTL_ERR_ACCES</code> | Permission denied. |
| <code>TNFCTL_ERR_ALLOCFAIL</code> | A memory allocation failure occurred. |

| | |
|--|---|
| TNFCTL_ERR_NOTDYNAMIC | The target is not a dynamic executable. |
| TNFCTL_ERR_NOLIBTNFPROBE | libtnfprobe.so.1 is not linked in the target process. |
| TNFCTL_ERR_FILENOTFOUND | The program is not found. |
| TNFCTL_ERR_INTERNAL | An internal error occurred. |
| The following error codes apply to <code>tnfctl_continue()</code> : | |
| TNFCTL_ERR_BADARG | Bad input argument. <i>hndl</i> is not a direct process probe control handle. |
| TNFCTL_ERR_INTERNAL | An internal error occurred. |
| TNFCTL_ERR_NOPROCESS | No such target process exists. |

EXAMPLES

EXAMPLE 1 The use of the `tnfctl_pid_open()` function.

These examples do not include any error-handling code. Only the initial example includes the declaration of the variables that is used for all the examples.

The following example shows how to preload `libtnfprobe.so.1` from the normal location and inherit the parent's environment.

```
const char\011*pgm;
char * const\011*argv;
tnfctl_handle_t\011*hndl, *new_hndl, *child_hndl;
tnfctl_errcode_t\011err;
char * const\011*envptr;
extern char\011**environ;
tnfctl_event_t\011evt;
int\011pid;

/* assuming argv has been allocated */
argv[0] = pgm;
/* set up rest of argument vector here */
err = tnfctl_exec_open(pgm, argv, NULL, NULL, NULL, &hndl);
```

This example shows how to preload two user-supplied libraries `libc_probe.so.1` and `libthread_probe.so.1`. They interpose on the corresponding `libc.so` and `libthread.so` interfaces and have probes for function entry and exit. `libtnfprobe.so.1` is preloaded from the normal location and the parent's environment is inherited.

```

/* assuming argv has been allocated */
argv[0] = pgm;
/* set up rest of argument vector here */
err = tnfctl_exec_open(pgm, argv, NULL, NULL,
    "libc_probe.so.1 libthread_probe.so.1", &hndl);

```

This example preloads an interposition library `libc_probe.so.1`, and specifies a different location from which to preload `libtnfprobe.so.1`.

```

/* assuming argv has been allocated */
argv[0] = pgm;
/* set up rest of argument vector here */
err = tnfctl_exec_open(pgm, argv, NULL, "/opt/SUNWXXX/lib",
    "libc_probe.so.1", &hndl);

```

To set up the environment explicitly for probe control to work, the target process must link `libtnfprobe.so.1`. If using `envp`, it is the caller's responsibility to do so.

```

/* assuming argv has been allocated */
argv[0] = pgm;
/* set up rest of argument vector here */
/* envptr set up to caller's needs */
err = tnfctl_exec_open(pgm, argv, envptr, NULL, NULL, &hndl);

```

Use this example to resume a process that does an `exec(2)` without controlling it.

```

err = tnfctl_continue(hndl, &evt, NULL);
switch (evt) {
case TNFCTL_EVENT_EXEC:
    /* let target process continue without control */
    err = tnfctl_close(hndl, TNFCTL_TARG_RESUME);
    ...
    break;
}

```

Alternatively, use the next example to control a process that does an `exec(2)`.

```

/*
 * assume the pid variable has been set by calling
 * tnfctl_trace_attrs_get()
 */
err = tnfctl_continue(hndl, &evt, NULL);
switch (evt) {
case TNFCTL_EVENT_EXEC:
    /* suspend the target process */
    err = tnfctl_close(hndl, TNFCTL_TARG_SUSPEND);
    /* re-open the exec'ed image */
    err = tnfctl_pid_open(pid, &new_hndl);
    /* new_hndl now controls the exec'ed image */
    ...
    break;
}

```

To let fork 'ed children continue without control, use NULL as the last argument to tnfctl_continue().

```
err = tnfctl_continue(hndl, &evt, NULL);
```

The next example is how to control child processes that `fork(2)` or `fork1(2)` create.

```

err = tnfctl_continue(hndl, &evt, &child_hndl);
switch (evt) {
case TNFCTL_EVENT_FORK:
    /* spawn a new thread or process to control child_hndl */
    ...
    break;
}

```

ATTRIBUTES

See [attributes\(5\)](#) for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| Availability | SUNWtnfc |
| MT Level | MT-Safe |

SEE ALSO

[ld\(1\)](#), [prex\(1\)](#), [proc\(1\)](#), [exec\(2\)](#), [execve\(2\)](#), [exit\(2\)](#), [fork\(2\)](#), [TNF_PROBE\(3X\)](#), [dlclose\(3X\)](#), [dlopen\(3X\)](#), [libtnfctl\(3X\)](#), [tnfctl_close\(3X\)](#), [tnfctl_internal_open\(3X\)](#), [tracing\(3X\)](#), [attributes\(5\)](#)

Programming Utilities Guide Linker and Libraries Guide

NOTES

After a `tnfctl_continue()` returns, a client should use `tnfctl_trace_attrs_get(3X)` to check the `trace_buf_state` member of the trace attributes and make sure that there is no internal error in the target.

| | |
|--------------------|---|
| NAME | tnfctl_probe_apply, tnfctl_probe_apply_ids – iterate over probes |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -ltnfctl [<i>library</i> ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_probe_apply(tnfctl_handle_t * <i>hndl</i>, tnfctl_probe_op_t <i>probe_op</i>, void * <i>clientdata</i>); tnfctl_errcode_t tnfctl_probe_apply_ids(tnfctl_handle_t * <i>hndl</i>, ulong_t <i>probe_count</i>, ulong_t * <i>probe_ids</i>, tnfctl_probe_op_t <i>probe_op</i>, void * <i>clientdata</i>);</pre> |
| DESCRIPTION | <p>tnfctl_probe_apply() is used to iterate over the probes controlled by <i>hndl</i>. For every probe, the <i>probe_op</i> function is called:</p> <pre>typedef tnfctl_errcode_t (*tnfctl_probe_op_t)(\011tnfctl_handle_t *hndl, \011tnfctl_probe_t *probe_hndl, \011void *clientdata);</pre> <p>Several predefined functions are available for use as <i>probe_op</i>. These functions are described in tnfctl_probe_state_get(3X).</p> <p>The <i>clientdata</i> supplied in tnfctl_probe_apply() is passed in as the last argument of <i>probe_op</i>. The <i>probe_hndl</i> in the probe operation function can be used to query or change the state of the probe. See tnfctl_probe_state_get(3X). The <i>probe_op</i> function should return TNFCTL_ERR_NONE upon success. It can also return an error code, which will cause tnfctl_probe_apply() to stop processing the rest of the probes and return with the same error code. Note that there are five (5) error codes reserved that the client can use for its own semantics. See ERRORS.</p> <p>The lifetime of <i>probe_hndl</i> is the same as the lifetime of <i>hndl</i>. It is good until <i>hndl</i> is closed by tnfctl_close(3X). Do not confuse a <i>probe_hndl</i> with <i>hndl</i>. The <i>probe_hndl</i> refers to a particular probe, while <i>hndl</i> refers to a process or the kernel. If <i>probe_hndl</i> is used in another libtnfctl(3X) interface, and it references a probe in a library that has been dynamically closed (see dlclose(3X)), then the error code TNFCTL_ERR_INVALIDPROBE will be returned by that interface.</p> |

tnfctl_probe_apply_ids() is very similar to **tnfctl_probe_apply()**. The difference is that *probe_op* is called only for probes that match a probe id specified in the array of integers referenced by *probe_ids*. The number of probe ids in the array should be specified in *probe_count*. Use **tnfctl_probe_state_get()** to get the *probe_id* that corresponds to the *probe_hndl*.

RETURN VALUES

tnfctl_probe_apply() and **tnfctl_probe_apply_ids()** return `TNFCTL_ERR_NONE` upon success.

ERRORS

The following errors apply to both **tnfctl_probe_apply()** and **tnfctl_probe_apply_ids()**:

| | |
|----------------------------------|-------------------------------|
| <code>TNFCTL_ERR_INTERNAL</code> | An internal error occurred. |
| <code>TNFCTL_ERR_USR1</code> | Error code reserved for user. |
| <code>TNFCTL_ERR_USR2</code> | Error code reserved for user. |
| <code>TNFCTL_ERR_USR3</code> | Error code reserved for user. |
| <code>TNFCTL_ERR_USR4</code> | Error code reserved for user. |
| <code>TNFCTL_ERR_USR5</code> | Error code reserved for user. |

tnfctl_probe_apply() and **tnfctl_probe_apply_ids()** also return any error returned by the callback function *probe_op*.

The following errors apply only to **tnfctl_probe_apply_ids()**:

| | |
|--------------------------------------|--|
| <code>TNFCTL_ERR_INVALIDPROBE</code> | The probe handle is no longer valid. For example, the probe is in a library that has been closed by <code>d1close(3X)</code> . |
|--------------------------------------|--|

EXAMPLES

EXAMPLE 1 Enabling probes.

To enable all probes:

```
tnfctl_probe_apply(hndl, tnfctl_probe_enable, NULL);
```

EXAMPLE 2 Disabling probes.

To disable the probes that match a certain pattern in the probe attribute string:

```
/* To disable all probes that contain the string "vm" */
tnfctl_probe_apply(hndl, select_disable, "vm");
static tnfctl_errcode_t
select_disable(tnfctl_handle_t *hndl, tnfctl_probe_t *probe_hndl,
void *client_data)
{
```

```

char *pattern = client_data;
tnfctl_probe_state_t probe_state;
tnfctl_probe_state_get(hndl, probe_hndl, &probe_state);
if (strstr(probe_state.attr_string, pattern)) {
    tnfctl_probe_disable(hndl, probe_hndl, NULL);
}
}

```

Note that these examples do not have any error handling code.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| Availability | SUNWtnfc |
| MT-Level | MT-Safe |

SEE ALSO

prex(1), **TNF_PROBE(3X)**, **dlclose(3X)**, **dlopen(3X)**, **libtnfctl(3X)**, **tnfctl_close(3X)**, **tnfctl_probe_state_get(3X)**, **tracing(3X)**, **tnf_kernel_probes(4)**, **attributes(5)**

Programming Utilities Guide Linker and Libraries Guide

| | | | |
|--------------------|--|-----------|--|
| NAME | tnfctl_probe_state_get, tnfctl_probe_enable, tnfctl_probe_disable, tnfctl_probe_trace, tnfctl_probe_untrace, tnfctl_probe_connect, tnfctl_probe_disconnect_all – interfaces to query and to change the state of a probe | | |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -ltnfctl [<i>library</i> ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_probe_state_get(tnfctl_handle_t * <i>hndl</i>, tnfctl_probe_t * <i>probe_hndl</i>, tnfctl_probe_state_t * <i>state</i>); tnfctl_errcode_t tnfctl_probe_enable(tnfctl_handle_t * <i>hndl</i>, tnfctl_probe_t * <i>probe_hndl</i>, void * <i>ignored</i>); tnfctl_errcode_t tnfctl_probe_disable(tnfctl_handle_t * <i>hndl</i>, tnfctl_probe_t * <i>probe_hndl</i>, void * <i>ignored</i>); tnfctl_errcode_t tnfctl_probe_trace(tnfctl_handle_t * <i>hndl</i>, tnfctl_probe_t * <i>probe_hndl</i>, void * <i>ignored</i>); tnfctl_errcode_t tnfctl_probe_untrace(tnfctl_handle_t * <i>hndl</i>, tnfctl_probe_t * <i>probe_hndl</i>, void * <i>ignored</i>); tnfctl_errcode_t tnfctl_probe_disconnect_all(tnfctl_handle_t * <i>hndl</i>, tnfctl_probe_t * <i>probe_hndl</i>, void * <i>ignored</i>); tnfctl_errcode_t tnfctl_probe_connect(tnfctl_handle_t * <i>hndl</i>, tnfctl_probe_t * <i>probe_hndl</i>, const char * <i>lib_base_name</i>, const char * <i>func_name</i>);</pre> | | |
| DESCRIPTION | <p>tnfctl_probe_state_get() returns the state of the probe specified by <i>probe_hndl</i> in the process or kernel specified by <i>hndl</i> . The user will pass these in to an apply iterator. The caller must also allocate <i>state</i> and pass in a pointer to it. The semantics of the individual members of <i>state</i> are:</p> <table border="0"> <tr> <td style="vertical-align: top; padding-right: 20px;"><i>id</i></td> <td>The unique integer assigned to this probe. This number does not change over the lifetime of this probe. A <i>probe_hndl</i> can be obtained by using the calls tnfctl_apply() , tnfctl_apply_ids() , or tnfctl_register_funcs() .</td> </tr> </table> | <i>id</i> | The unique integer assigned to this probe. This number does not change over the lifetime of this probe. A <i>probe_hndl</i> can be obtained by using the calls tnfctl_apply() , tnfctl_apply_ids() , or tnfctl_register_funcs() . |
| <i>id</i> | The unique integer assigned to this probe. This number does not change over the lifetime of this probe. A <i>probe_hndl</i> can be obtained by using the calls tnfctl_apply() , tnfctl_apply_ids() , or tnfctl_register_funcs() . | | |

| | |
|-------------|--|
| attr_string | <p>A string that consists of <i>attribute value</i> pairs separated by semicolons. For the syntax of this string, see the syntax of the <code>detail</code> argument of the <code>TNF_PROBE(3X)</code> macro. The attributes <code>name</code>, <code>slots</code>, <code>keys</code>, <code>file</code>, and <code>line</code> are defined for every probe. Additional user-defined attributes can be added by using the <code>detail</code> argument of the <code>TNF_PROBE(3X)</code> macro. An example of <code>attr_string</code> follows:</p> <pre>"name pageout;slots vnode pages_pageout ; keys vm pageio io;file vm.c;line 25;"</pre> |
| enabled | <p><code>B_TRUE</code> if the probe is enabled, or <code>B_FALSE</code> if the probe is disabled. Probes are disabled by default. Use <code>tnfctl_probe_enable()</code> or <code>tnfctl_probe_disable()</code> to change this state.</p> |
| traced | <p><code>B_TRUE</code> if the probe is traced, or <code>B_FALSE</code> if the probe is not traced. Probes in user processes are traced by default. Kernel probes are untraced by default. Use <code>tnfctl_probe_trace()</code> or <code>tnfctl_probe_untrace()</code> to change this state.</p> |
| new_probe | <p><code>B_TRUE</code> if this is a new probe brought in since the last change in libraries. See <code>dlopen(3X)</code> or <code>dlclose(3X)</code>. Otherwise, the value of <code>new_probe</code> will be <code>B_FALSE</code>. This field is not meaningful for kernel probe control.</p> |
| obj_name | <p>The name of the shared object or executable in which the probe is located. This string can be freed, so the client should make a copy of the string if it needs to be saved for use by other <code>libtnfctl</code> interfaces. In kernel mode, this string is always <code>NULL</code>.</p> |
| func_names | <p>A null-terminated array of pointers to strings that contain the names of functions connected to this probe. Whenever an enabled probe is encountered at runtime, these functions are executed. This array also will be freed by the library when the state of the probe changes. Use <code>tnfctl_probe_connect()</code> or <code>tnfctl_probe_disconnect_all()</code> to change this state.</p> |

`func_addr` A null-terminated array of pointers to addresses of functions in the target image connected to this probe. This array also will be freed by the library when the state of the probe changes.

`client_registered_data` Data that was registered by the client for this probe by the creator function in `tnfctl_register_funcs(3X)`.

`tnfctl_probe_enable()`, `tnfctl_probe_disable()`, `tnfctl_probe_trace()`, `tnfctl_probe_untrace()`, and `tnfctl_probe_disconnect_all()` ignore the last argument. This convenient feature permits these functions to be used in the `probe_op` field of `tnfctl_probe_apply(3X)` and `tnfctl_probe_apply_ids(3X)`. `tnfctl_probe_enable()` enables the probe specified by `probe_hdl`. This is the master switch on a probe. A probe does not perform any action until it is enabled.

`tnfctl_probe_disable()` disables the probe specified by `probe_hdl`.

`tnfctl_probe_trace()` turns on tracing for the probe specified by `probe_hdl`. Probes emit a trace record only if the probe is traced.

`tnfctl_probe_untrace()` turns off tracing for the probe specified by `probe_hdl`. This is useful if you want to connect probe functions to a probe without tracing it.

`tnfctl_probe_connect()` connects the function `func_name` which exists in the library `lib_base_name`, to the probe specified by `probe_hdl`.

`tnfctl_probe_connect()` returns an error code if used on a kernel tnfctl handle. `lib_base_name` is the base name (not a path) of the library. If it is `NULL`, and multiple functions in the target process match `func_name`, one of the matching functions is chosen arbitrarily. A probe function is a function that is in the target's address space and is written to a certain specification. The specification is not currently published.

`tnf_probe_debug()` is one function exported by `libtnfprobe.so.1` and is the debug function that `prex(1)` uses. When the debug function is executed, it prints out the probe arguments and the value of the `sunw%debug` attribute of the probe to `stderr`.

`tnfctl_probe_disconnect_all()` disconnects all probe functions from the probe specified by `probe_hdl`.

Note that no `libtnfctl` call returns a probe handle (`tnfctl_probe_t`), yet each of the routines described here takes a `probe_hdl` as an argument. These routines may be used by passing them to one of the `tnfctl_probe_apply(3X)` iterators as the "op" argument. Alternatively,

probe handles may be obtained and saved by a user's "op" function, and they can be passed later as the *probe_hdl* argument when using any of the functions described here.

RETURN VALUES

tnfctl_probe_state_get(), tnfctl_probe_enable(), tnfctl_probe_disable(), tnfctl_probe_trace(), tnfctl_probe_untrace(), **tnfctl_probe_disconnect_all()** and **tnfctl_probe_connect()** return TNFCTL_ERR_NONE upon success.

ERRORS

The following error codes apply to **tnfctl_probe_state_get()** :

TNFCTL_ERR_INVALIDPROBE *probe_hdl* is no longer valid. The library that the probe was in could have been dynamically closed by **d1close(3X)** .

The following error codes apply to **tnfctl_probe_enable()** , **tnfctl_probe_disable()** , **tnfctl_probe_trace()** , **tnfctl_probe_untrace()** , and **tnfctl_probe_disconnect_all()**

TNFCTL_ERR_INVALIDPROBE *probe_hdl* is no longer valid. The library that the probe was in could have been dynamically closed by **d1close(3X)** .

TNFCTL_ERR_BUFBROKEN Cannot do probe operations because tracing is broken in the target.

TNFCTL_ERR_NOBUF Cannot do probe operations until a buffer is allocated. See **tnfctl_buffer_alloc(3X)** . This error code does not apply to kernel probe control.

The following error codes apply to **tnfctl_probe_connect()** :

TNFCTL_ERR_INVALIDPROBE *probe_hdl* is no longer valid. The library that the probe was in could have been dynamically closed by **d1close(3X)** .

TNFCTL_ERR_BADARG The handle is a kernel handle, or *func_name* could not be found.

TNFCTL_ERR_BUFBROKEN Cannot do probe operations because tracing is broken in the target.

TNFCTL_ERR_NOBUF Cannot do probe operations until a buffer is allocated. See **tnfctl_buffer_alloc(3X)** .

ATTRIBUTES

See **attributes(5)** for description of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| Availability | SUNWtnfc |
| MT Level | MT-Safe |

SEE ALSO

prex(1), **TNF_PROBE(3X)**, **libtnfctl(3X)**, **tnfctl_check_libs(3X)**,
tnfctl_continue(3X), **tnfctl_probe_apply(3X)**,
tnfctl_probe_apply_ids(3X), **tracing(3X)**, **tnf_kernel_probes(4)**,
attributes(5)

Programming Utilities Guide

| NAME | tnfctl_register_funcs - register callbacks for probe creation and destruction | | | | | | |
|----------------------|---|----------------|-----------------|--------------|----------|----------|---------|
| SYNOPSIS | <pre>cc [flag ...] file ... -ltnfctl [library ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_register_funcs(tnfctl_handle_t *hndl, void * (*create_func)(tnfctl_handle_t *,tnfctl_probe_t*), void (*destroy_func)(void *));</pre> | | | | | | |
| DESCRIPTION | <p>The function tnfctl_register_funcs() is used to store client-specific data on a per-probe basis. It registers a creator and a destructor function with <i>hndl</i>, either of which can be NULL. The creator function is called for every probe that currently exists in <i>hndl</i>. Every time a new probe is discovered, that is brought in by dlopen(3X), <i>create_func</i> is called.</p> <p>The return value of the creator function is stored as part of the probe state and can be retrieved by tnfctl_probe_state_get(3X) in the member field <i>client_registered_data</i>.</p> <p><i>destroy_func</i> is called for every probe handle that is freed. This does not necessarily happen at the time dldclose(3X) frees the shared object. The probe handles are freed only when <i>hndl</i> is closed by tnfctl_close(3X). If tnfctl_register_funcs() is called a second time for the same <i>hndl</i>, then the previously registered destructor function is called first for all of the probes.</p> | | | | | | |
| RETURN VALUES | tnfctl_register_funcs() returns TNFCTL_ERR_NONE upon success. | | | | | | |
| ERRORS | TNFCTL_ERR_INTERNAL An internal error occurred. | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>Availability</td> <td>SUNWtnfc</td> </tr> <tr> <td>MT Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | Availability | SUNWtnfc | MT Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| Availability | SUNWtnfc | | | | | | |
| MT Level | MT-Safe | | | | | | |
| SEE ALSO | <pre>prex(1), TNF_PROBE(3X), dlclose(3X), dlopen(3X), libtnfctl(3X), tnfctl_close(3X), tnfctl_probe_state_get(3X), tracing(3X), tnf_kernel_probes(4), attributes(5)</pre> <p><i>Programming Utilities Guide Linker and Libraries Guide</i></p> | | | | | | |

- NAME** | tnfctl_strerror – map a tnfctl error code to a string
- SYNOPSIS** |

```
cc [ flag ... ] file ... -ltnfctl [ library ... ]
#include <tnf/tnfctl.h>

const char * tnfctl_strerror(tnfctl_errcode_t errcode);
```
- DESCRIPTION** | **tnfctl_strerror()** maps the error number in *errcode* to an error message string, and it returns a pointer to that string. The returned string should not be overwritten or freed.
- ERRORS** | **tnfctl_strerror()** returns the string "unknown libtnfctl.so error code" if the error number is not within the legal range.
- ATTRIBUTES** | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| Availability | SUNWtnfc |
| MT Level | MT-Safe |

- SEE ALSO** | **prex(1)**, **TNF_PROBE(3X)**, **libtnfctl(3X)**, **tracing(3X)**, **attributes(5)**
Programming Utilities Guide

| | | | | | | | |
|------------------------------|--|-----------------------|---|------------------------------|---|-----------------------------|--|
| NAME | tnfctl_trace_attrs_get – get the trace attributes from a tnfctl handle | | | | | | |
| SYNOPSIS | <pre>cc [flag ...] file ... -ltnfctl [library ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_trace_attrs_get(tnfctl_handle_t *hndl, tnfctl_trace_attrs_t *attrs);</pre> | | | | | | |
| DESCRIPTION | <p>tnfctl_trace_attrs_get() returns the trace attributes associated with <i>hndl</i> in <i>attrs</i>. The trace attributes can be changed by some of the other interfaces in libtnfctl(3X). It is the client's responsibility to use tnfctl_trace_attrs_get() to get the new trace attributes after use of interfaces that change them. Typically, a client will use tnfctl_trace_attrs_get() after a call to tnfctl_continue(3X) in order to make sure that tracing is still working. See the discussion of <i>trace_buf_state</i> that follows.</p> <p>Trace attributes are represented by the struct <code>tnfctl_trace_attrs</code> structure defined in <code><tnf/tnfctl.h></code>:</p> <pre>struct tnfctl_trace_attrs { pid_t targ_pid; /* not kernel mode */ const char *trace_file_name; /* not kernel mode */ size_t trace_buf_size; size_t trace_min_size; tnfctl_bufstate_t trace_buf_state; boolean_t trace_state; boolean_t filter_state; /* kernel mode only */ long pad; };</pre> <p>The semantics of the individual members of <i>attrs</i> are:</p> <table border="0"> <tr> <td style="padding-right: 20px;"><code>targ_pid</code></td> <td>The process id of the target process. This is not valid for kernel tracing.</td> </tr> <tr> <td style="padding-right: 20px;"><code>trace_file_name</code></td> <td>The name of the trace file to which the target writes. <code>trace_file_name</code> will be NULL if no trace file exists or if kernel tracing is implemented. This pointer should not be used after calling other <code>libtnfctl</code> interfaces. The client should copy this string if it should be saved for the use of other <code>libtnfctl</code> interfaces.</td> </tr> <tr> <td style="padding-right: 20px;"><code>trace_buf_size</code></td> <td>The size of the trace buffer or file in bytes.</td> </tr> </table> | <code>targ_pid</code> | The process id of the target process. This is not valid for kernel tracing. | <code>trace_file_name</code> | The name of the trace file to which the target writes. <code>trace_file_name</code> will be NULL if no trace file exists or if kernel tracing is implemented. This pointer should not be used after calling other <code>libtnfctl</code> interfaces. The client should copy this string if it should be saved for the use of other <code>libtnfctl</code> interfaces. | <code>trace_buf_size</code> | The size of the trace buffer or file in bytes. |
| <code>targ_pid</code> | The process id of the target process. This is not valid for kernel tracing. | | | | | | |
| <code>trace_file_name</code> | The name of the trace file to which the target writes. <code>trace_file_name</code> will be NULL if no trace file exists or if kernel tracing is implemented. This pointer should not be used after calling other <code>libtnfctl</code> interfaces. The client should copy this string if it should be saved for the use of other <code>libtnfctl</code> interfaces. | | | | | | |
| <code>trace_buf_size</code> | The size of the trace buffer or file in bytes. | | | | | | |

| | |
|-----------------|---|
| trace_min_size | The minimum size in bytes of the trace buffer that can be allocated by using the <code>tnfctl_buffer_alloc(3X)</code> interface. |
| trace_buf_state | The state of the trace buffer. <code>TNFCTL_BUF_OK</code> indicates that a trace buffer has been allocated. <code>TNFCTL_BUF_NONE</code> indicates that no buffer has been allocated. <code>TNFCTL_BUF_BROKEN</code> indicates that there is an internal error in the target for tracing. The target will continue to run correctly, but no trace records will be written. To fix tracing, restart the process. For kernel tracing, deallocate the existing buffer with <code>tnfctl_buffer_dealloc(3X)</code> and allocate a new one with <code>tnfctl_buffer_alloc(3X)</code> . |
| trace_state | The global tracing state of the target. Probes that are enabled will not write out data unless this state is on. This state is off by default for the kernel and can be changed by <code>tnfctl_trace_state_set(3X)</code> . For a process, this state is on by default and can only be changed by <code>tnf_process_disable(3X)</code> and <code>tnf_process_enable(3X)</code> . |
| filter_state | The state of process filtering. For kernel probe control, it is possible to select a set of processes for which probes are enabled. See <code>tnfctl_filter_list_get(3X)</code> , <code>tnfctl_filter_list_add(3X)</code> , and <code>tnfctl_filter_list_delete(3X)</code> . No trace output will be written when other processes traverse these probe points. By default process filtering is off, and all processes cause the generation of trace records when they hit an enabled probe. Use <code>tnfctl_filter_state_set(3X)</code> to change the filter state. |

RETURN VALUES

`tnfctl_trace_attrs_get()` returns `TNFCTL_ERR_NONE` upon success.

ERRORS

The following error codes apply to `tnfctl_trace_attrs_get()`

`TNFCTL_ERR_INTERNAL`

An internal error occurred.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| Availability | SUNWtnfc |
| MT Level | MT-Safe |

SEE ALSO

prex(1), **TNF_PROBE(3X)**, **libtnfctl(3X)**, **tnfctl_buffer_alloc(3X)**,
tnfctl_continue(3X), **tnfctl_filter_list_get (3X)**,
tnf_process_disable(3X), **tracing(3X)**, **attributes(5)**

Programming Utilities Guide

| | |
|--------------------|--|
| NAME | tnfctl_trace_state_set, tnfctl_filter_state_set, tnfctl_filter_list_get, tnfctl_filter_list_add, tnfctl_filter_list_delete – control kernel tracing and process filtering |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -ltnfctl [<i>library</i> ...] #include <tnf/tnfctl.h> tnfctl_errcode_t tnfctl_trace_state_set(tnfctl_handle_t * <i>hndl</i>, boolean_t <i>trace_state</i>); tnfctl_errcode_t tnfctl_filter_state_set(tnfctl_handle_t * <i>hndl</i>, boolean_t <i>filter_state</i>); tnfctl_errcode_t tnfctl_filter_list_get(tnfctl_handle_t * <i>hndl</i>, pid_t ** <i>pid_list</i>, int * <i>pid_count</i>); tnfctl_errcode_t tnfctl_filter_list_add(tnfctl_handle_t * <i>hndl</i>, pid_t <i>pid_to_add</i>); tnfctl_errcode_t tnfctl_filter_list_delete(tnfctl_handle_t * <i>hndl</i>, pid_t <i>pid_to_delete</i>);</pre> |
| DESCRIPTION | <p>The interfaces to control kernel tracing and process filtering are used only with kernel handles, handles created by <code>tnfctl_kernel_open(3X)</code>. These interfaces are used to change the tracing and filter states for kernel tracing.</p> <p><code>tnfctl_trace_state_set()</code> sets the kernel global tracing state to "on" if <code>trace_state</code> is <code>B_TRUE</code>, or to "off" if <code>trace_state</code> is <code>B_FALSE</code>. For the kernel, <code>trace_state</code> is off by default. Probes that are enabled will not write out data unless this state is on. Use <code>tnfctl_trace_attrs_get(3X)</code> to retrieve the current tracing state.</p> <p><code>tnfctl_filter_state_set()</code> sets the kernel process filtering state to "on" if <code>filter_state</code> is <code>B_TRUE</code>, or to "off" if <code>filter_state</code> is <code>B_FALSE</code>. <code>filter_state</code> is off by default. If it is on, only probe points encountered by processes in the process filter set by <code>tnfctl_filter_list_add()</code> will generate trace points. Use <code>tnfctl_trace_attrs_get(3X)</code> to retrieve the current process filtering state.</p> <p><code>tnfctl_filter_list_get()</code> returns the process filter list as an array in <code>pid_list</code>. The count of elements in the process filter list is returned in <code>pid_count</code>. The caller should use <code>free(3C)</code> to free memory allocated for the array <code>pid_list</code>.</p> |

RETURN VALUES

tnfctl_filter_list_add() adds *pid_to_add* to the process filter list. The process filter list is maintained even when the process filtering state is off, but it has no effect unless the process filtering state is on.

tnfctl_filter_list_delete() deletes *pid_to_delete* from the process filter list. It returns an error if the process does not exist or is not in the filter list.

The interfaces **tnfctl_trace_state_set()**, **tnfctl_filter_state_set()**, **tnfctl_filter_list_add()**, **tnfctl_filter_list_delete()**, and **tnfctl_filter_list_get()** return `TNFCTL_ERR_NONE` upon success.

ERRORS

The following error codes apply to `tnfctl_trace_state_set`:

`TNFCTL_ERR_BADARG` The handle is not a kernel handle.

`TNFCTL_ERR_NOBUF` Cannot turn on tracing without a buffer being allocated.

`TNFCTL_ERR_BUFBROKEN` Tracing is broken in the target.

`TNFCTL_ERR_INTERNAL` An internal error occurred.

The following error codes apply to `tnfctl_filter_state_set`:

`TNFCTL_ERR_BADARG` The handle is not a kernel handle.

`TNFCTL_ERR_INTERNAL` An internal error occurred.

The following error codes apply to `tnfctl_filter_list_add`:

`TNFCTL_ERR_BADARG` The handle is not a kernel handle.

`TNFCTL_ERR_NOPROCESS` No such process exists.

`TNFCTL_ERR_ALLOCFAIL` A memory allocation failure occurred.

`TNFCTL_ERR_INTERNAL` An internal error occurred.

The following error codes apply to `tnfctl_filter_list_delete`:

`TNFCTL_ERR_BADARG` The handle is not a kernel handle.

`TNFCTL_ERR_NOPROCESS` No such process exists.

`TNFCTL_ERR_INTERNAL` An internal error occurred.

The following error codes apply to `tnfctl_filter_list_get`:

`TNFCTL_ERR_BADARG` The handle is not a kernel handle.

`TNFCTL_ERR_ALLOCFAIL` A memory allocation failure occurred.

`TNFCTL_ERR_INTERNAL` An internal error occurred.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| Availability | SUNWtnfc |
| MT Level | MT-Safe |

SEE ALSO

prex(1), **TNF_PROBE(3X)**, **free(3C)**, **libtnfctl(3X)**,
tnfctl_kernel_open(3X), **tnfctl_trace_attrs_get(3X)**,
tracing(3X), **tnf_kernel_probes(4)**, **attributes(5)**

Programming Utilities Guide

| | |
|--------------------|--|
| NAME | TNF_DECLARE_RECORD, TNF_DEFINE_RECORD_1, TNF_DEFINE_RECORD_2, TNF_DEFINE_RECORD_3, TNF_DEFINE_RECORD_4, TNF_DEFINE_RECORD_5 – TNF type extension interface for probes |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] <i>file</i> ...[-ltnfprobe] [<i>library</i> ...] #include <tnf/probe.h> TNF_DECLARE_RECORD(<i>c_type</i>, <i>tnf_type</i>); TNF_DEFINE_RECORD_1(<i>c_type</i>, <i>tnf_type</i>, <i>tnf_member_type_1</i>, <i>c_member_name_1</i>); TNF_DEFINE_RECORD_2(<i>c_type</i>, <i>tnf_type</i>, <i>tnf_member_type_1</i>, <i>c_member_name_1</i>, <i>tnf_member_type_2</i>, <i>c_member_name_2</i>); TNF_DEFINE_RECORD_3(<i>c_type</i>, <i>tnf_type</i>, <i>tnf_member_type_1</i>, <i>c_member_name_1</i>, <i>tnf_member_type_2</i>, <i>c_member_name_2</i>, <i>tnf_member_type_3</i>, <i>c_member_name_3</i>); TNF_DEFINE_RECORD_4(<i>c_type</i>, <i>tnf_type</i>, <i>tnf_member_type_1</i>, <i>c_member_name_1</i>, <i>tnf_member_type_2</i>, <i>c_member_name_2</i>, <i>tnf_member_type_3</i>, <i>c_member_name_3</i>, <i>tnf_member_type_4</i>, <i>c_member_name_4</i>); TNF_DEFINE_RECORD_5(<i>c_type</i>, <i>tnf_type</i>, <i>tnf_member_type_1</i>, <i>c_member_name_1</i>, <i>tnf_member_type_2</i>, <i>c_member_name_2</i>, <i>tnf_member_type_3</i>, <i>c_member_name_3</i>, <i>tnf_member_type_4</i>, <i>c_member_name_4</i>, <i>tnf_member_type_5</i>, <i>c_member_name_5</i>); </pre> |
| DESCRIPTION | <p>This macro interface is used to extend the TNF (Trace Normal Form) types that can be used in TNF_PROBE(3X) .</p> <p>There should be only one TNF_DECLARE_RECORD and one TNF_DEFINE_RECORD per new type being defined. The TNF_DECLARE_RECORD should precede the TNF_DEFINE_RECORD . It can be in a header file that multiple source files share if those source files need to use the <i>tnf_type</i> being defined. The TNF_DEFINE_RECORD should only appear in one of the source files.</p> <p>The TNF_DEFINE_RECORD macro interface defines a function as well as a couple of data structures. Hence, this interface has to be used in a source file (.c or .cc file) at file scope and not inside a function.</p> |

Note that there is no semicolon after the `TNF_DEFINE_RECORD` interface. Having one will generate a compiler warning.

Compiling with the preprocessor option `-DNPROBE` (see `cc(1B)`), or with the preprocessor control statement `#define NPROBE` ahead of the `#include <tnf/probe.h>` statement, will stop the TNF type extension code from being compiled into the program.

c_type *c_type* must be a C struct type. It is the template from which the new *tnf_type* is being created. Not all elements of the C struct need be provided in the TNF type being defined.

tnf_type *tnf_type* is the name being given to the newly created type. Use of this interface uses the name space prefixed by *tnf_type*. So, if a new type called "xxx_type" is defined by a library, then the library should not use "xxx_type" as a prefix in any other symbols it defines. The policy on managing the type name space is the same as managing any other name space in a library i.e., prefix any new TNF types by the unique prefix that the rest of the symbols in the library use. This would prevent name space collisions when linking multiple libraries that define new TNF types. For example, if a library `libpalloc.so` uses the prefix "pal" for all symbols it defines, then it should also use the prefix "pal" for all new TNF types being defined.

tnf_member_type_n *tnf_member_type_n* is the TNF type of the *n* th provided member of the C structure.

tnf_member_name_n *tnf_member_name_n* is the name of the *n* th provided member of the C structure.

EXAMPLES

EXAMPLE 1 Defining and using a TNF type.

This example shows how a new TNF type is defined and used in a probe. This code is assumed to be part of a fictitious library called "libpalloc.so" which uses the prefix "pal" for all it's symbols.

```
#include <tnf/probe.h>
typedef struct pal_header {
    long    size;
    char *  descriptor;
    struct pal_header *next;
} pal_header_t;
TNF_DECLARE_RECORD(pal_header_t, pal_tnf_header);
TNF_DEFINE_RECORD_2(pal_header_t, pal_tnf_header,
                   tnf_long,    size,
                   tnf_string, descriptor)

/*
 * Note: name space prefixed by pal_tnf_header should not be used by this
 *       client anymore.
 */
void
pal_free(pal_header_t *header_p)
```

```

{
    int state;
    TNF_PROBE_2(pal_free_start, "palloc pal_free",
               "sunw%debug entering pal_free",
               tnf_long,      state_var, state,
               pal_tnf_header, header_var, header_p);
    . . .
}

```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| Availability | SUNWtnfd |
| MT-Level | MT-Safe |

SEE ALSO

prex(1), **tnfdump(1)**, **TNF_PROBE(3X)**, **tnf_process_disable(3X)**, **attributes(5)**

NOTES

It is possible to make a *tnf_type* definition be recursive or mutually recursive e.g. a structure that uses the "next" field to point to itself (a linked list). If such a structure is sent in to a **TNF_PROBE(3X)**, then the entire linked list will be logged to the trace file (until the "next" field is NULL). But, if the list is circular, it will result in an infinite loop. To break the recursion, either don't include the "next" field in the *tnf_type*, or define the type of the "next" member as *tnf_opaque*.

| | |
|-----------------|---|
| NAME | TNF_PROBE, TNF_PROBE_0, TNF_PROBE_1, TNF_PROBE_2, TNF_PROBE_3, TNF_PROBE_4, TNF_PROBE_5, TNF_PROBE_0_DEBUG, TNF_PROBE_1_DEBUG, TNF_PROBE_2_DEBUG, TNF_PROBE_3_DEBUG, TNF_PROBE_4_DEBUG, TNF_PROBE_5_DEBUG, TNF_DEBUG – probe insertion interface |
| SYNOPSIS | <pre> cc [<i>flag</i> ...] [-DTNF_DEBUG] <i>file</i> ... [-ltnfprobe] [<i>library</i> ...] #include <tnf/probe.h> TNF_PROBE_0(<i>name, keys, detail</i>); TNF_PROBE_1(<i>name, keys, detail, arg_type_1, arg_name_1, arg_value_1</i>); TNF_PROBE_2(<i>name, keys, detail, arg_type_1, arg_name_1, arg_value_1, arg_type_2, arg_name_2, arg_value_2</i>); TNF_PROBE_3(<i>name, keys, detail, arg_type_1, arg_name_1, arg_value_1, arg_type_2, arg_name_2, arg_value_2, arg_type_3, arg_name_3, arg_value_3</i>); TNF_PROBE_4(<i>name, keys, detail, arg_type_1, arg_name_1, arg_value_1, arg_type_2, arg_name_2, arg_value_2, arg_type_3, arg_name_3, arg_value_3, arg_type_4, arg_name_4, arg_value_4</i>); TNF_PROBE_5(<i>name, keys, detail, arg_type_1, arg_name_1, arg_value_1, arg_type_2, arg_name_2, arg_value_2, arg_type_3, arg_name_3, arg_value_3, arg_type_4, arg_name_4, arg_value_4, arg_type_5, arg_name_5, arg_value_5</i>); TNF_PROBE_0_DEBUG(<i>name, keys, detail</i>); TNF_PROBE_1_DEBUG(<i>name, keys, detail, arg_type_1, arg_name_1, arg_value_1</i>); TNF_PROBE_2_DEBUG(<i>name, keys, detail, arg_type_1, arg_name_1, arg_value_1, arg_type_2, arg_name_2, arg_value_2</i>); TNF_PROBE_3_DEBUG(<i>name, keys, detail, arg_type_1, arg_name_1, arg_value_1, arg_type_2, arg_name_2, arg_value_2, arg_type_3, arg_name_3, arg_value_3</i>); </pre> |

```
TNF_PROBE_4_DEBUG(name, keys, detail, arg_type_1, arg_name_1, arg_value_1, arg_type_2,
arg_name_2, arg_value_2, arg_type_3, arg_name_3, arg_value_3, arg_type_4, arg_name_4,
arg_value_4);
```

```
TNF_PROBE_5_DEBUG(name, keys, detail, arg_type_1, arg_name_1, arg_value_1, arg_type_2,
arg_name_2, arg_value_2, arg_type_3, arg_name_3, arg_value_3, arg_type_4, arg_name_4,
arg_value_4, arg_type_5, arg_name_5, arg_value_5);
```

DESCRIPTION

This macro interface is used to insert probes into C or C++ code for tracing. See `tracing(3X)` for a discussion of the Solaris tracing architecture, including example source code that uses it.

You can place probes anywhere in C and C++ programs including `.init` sections, `.fini` sections, multi-threaded code, shared objects, and shared objects opened by `dlopen(3X)`. Use probes to generate trace data for performance analysis or to write debugging output to `stderr`. Probes are controlled at runtime by `prex(1)`.

The trace data is logged to a trace file in Trace Normal Form (TNF). The interface for the user to specify the name and size of the trace file is described in `prex(1)`. Think of the trace file as the least recently used circular buffer. Once the file has been filled, newer events will overwrite the older ones.

Use `TNF_PROBE_0` through `TNF_PROBE_5` to create production probes. These probes are compiled in by default. Developers are encouraged to embed such probes strategically, and to leave them compiled within production software. Such probes facilitate on-site analysis of the software.

Use `TNF_PROBE_0_DEBUG` through `TNF_PROBE_5_DEBUG` to create debug probes. These probes are compiled out by default. If you compile the program with the preprocessor option `-DTNF_DEBUG` (see `cc(1B)`), or with the preprocessor control statement `#define TNF_DEBUG` ahead of the `#include <tnf/probe.h>` statement, the debug probes will be compiled into the program. When compiled in, debug probes differ in only one way from the equivalent production probes. They contain an additional "debug" attribute which may be used to distinguish them from production probes at runtime, for example, when using `prex()`. Developers are encouraged to embed any number of probes for debugging purposes. Disabled probes have such a small runtime overhead that even large numbers of them do not make a significant impact.

If you compile with the preprocessor option `-DNPROBE` (see `cc(1B)`), or place the preprocessor control statement `#define NPROBE` ahead of the `#include <tnf/probe.h>` statement, no probes will be compiled into the program.

name

The *name* of the probe should follow the syntax guidelines for identifiers in ANSI C. The use of *name* declares it, hence no separate declaration is

necessary. This is a block scope declaration, so it does not affect the name space of the program.

keys *keys* is a string of space-separated keywords that specify the groups that the probe belongs to. Semicolons, single quotation marks, and the equal character (=) are not allowed in this string. If any of the groups are enabled, the probe is enabled. *keys* cannot be a variable. It must be a string constant.

detail *detail* is a string that consists of <attribute> <value> pairs that are each separated by a semicolon. The first word (up to the space) is considered to be the attribute and the rest of the string (up to the semicolon) is considered the value. Single quotation marks are used to denote a string value. Besides quotation marks, spaces separate multiple values. The value is optional. Although semicolons or single quotation marks generally are not allowed within either the attribute or the value, when text with embedded spaces is meant to denote a single value, use single quotes surrounding this text.

Use *detail* for one of two reasons. First, use *detail* to supply an attribute that a user can type into `prex(1)` to select probes. For example, if a user defines an attribute called `color`, then `prex(1)` can select probes based on the value of `color`. Second, use *detail* to annotate a probe with a string that is written out to a trace file only once. `prex(1)` uses spaces to tokenize the value when searching for a match. Spaces around the semicolon delimiter are allowed. *detail* cannot be a variable; it must be a string constant. For example, the *detail* string:

```
"XYZ%debug 'entering function A'; XYZ%exception 'no file'; XYZ%func_entry; XYZ%color red blue"
```

consists of 4 units:

| attribute | value | values that prex matches on |
|----------------|-----------------------|-----------------------------|
| XYZ%debug | 'entering function A' | 'entering function A' |
| XYZ%exception | 'no file' | 'no file' |
| XYZ%func_entry | ./.* | (regular expression) |
| XYZ%color | red blue | red <or> blue |

Attribute names must be prefixed by the vendor stock symbol followed by the '%' character. This avoids conflicts in the attribute name space. All attributes that do not have a '%' character are reserved. The following attributes are predefined:

| | |
|-----------|--|
| attribute | semantics |
| name | name of probe |
| keys | keys of the probe (value is space-separated tokens) |
| file | file name of the probe |
| line | line number of the probe |
| slots | slot names of the probe event (<i>arg_name_n</i>) |
| object | the executable or shared object that this probe is in. |
| debug | distinguishes debug probes from production probes |

arg_type_n

This is the type of the *n* th argument. The following are predefined TNF types:

| | |
|---------------|---|
| tnf type | associated C type (and semantics) |
| tnf_int | int |
| tnf_uint | unsigned int |
| tnf_long | long |
| tnf_ulong | unsigned long |
| tnf_longlong | long long (if implemented in compilation system) |
| tnf_ulonglong | unsigned long long (if implemented in compilation system) |
| tnf_float | float |
| tnf_double | double |
| tnf_string | char * |
| tnf_opaque | void * |

To define new TNF types that are records consisting of the predefined TNF types or references to other user defined types, use the interface specified in **TNF_DECLARE_RECORD(3X)** .

arg_name_n

arg_name_n is the name that the user associates with the *n* th argument. Do not place quotation marks around *arg_name_n* . Follow the syntax guidelines for

identifiers in ANSI C. The string version of *arg_name_n* is stored for every probe and can be accessed as the attribute "slots".

arg_value_n *arg_value_n* is evaluated to yield a value to be included in the trace file. A read access is done on any variables that are in mentioned in *arg_value_n*. In a multi-threaded program, it is the user's responsibility to place locks around the TNF_PROBE macro if *arg_value_n* contains a variable that should be read protected.

EXAMPLES EXAMPLE 1 `tracing(3X)`.

See `tracing(3X)` for complete examples showing debug and production probes in source code.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| Availability | SUNWtnfd |
| MT Level | MT-Safe |

SEE ALSO `cc(1B)`, `ld(1)`, `prex(1)`, `tnfdump(1)`, `libthread(3T)`, `libtnfctl(3X)`, `TNF_DECLARE_RECORD(3X)`, `dlopen(3X)`, `libtnfctl(3X)`, `tnf_process_disable(3X)`, `tracing(3X)`, `attributes(5)`

Programming Utilities Guide

NOTES If attaching to a running program with `prex(1)` to control the probes, compile the program with `-ltnfprobe` or start the program with the environment variable `LD_PRELOAD` set to `libtnfprobe.so.1`. See `ld(1)`. If `libtnfprobe` is explicitly linked into the program, it must be before `libthread` on the link line.

| NAME | tnf_process_disable, tnf_process_enable, tnf_thread_disable, tnf_thread_enable - probe control internal interface | | | | | | |
|--------------------|--|----------------|-----------------|--------------|----------|----------|---------|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -ltnfprobe [<i>library</i> ...] #include <tnf/probe.h> void tnf_process_disable(void); void tnf_process_enable(void); void tnf_thread_disable(void); void tnf_thread_enable(void);</pre> | | | | | | |
| DESCRIPTION | <p>There are three levels of granularity for controlling tracing and probe functions (called probing from here on) — probing for the entire process, a particular thread, and the probe itself can be disabled/enabled. The first two (process and thread) are controlled by this interface. The probe is controlled via the application <code>prex(1)</code>.</p> <p>tnf_process_disable() turns off probing for the process. The default process state is to have probing enabled. tnf_process_enable() turns on probing for the process.</p> <p>tnf_thread_disable() turns off probing for the currently running thread. Threads are "born" or created with this state enabled. tnf_thread_enable() turns on probing for the currently running thread. If the program is a non-threaded program, these two thread interfaces disable or enable probing for the process.</p> | | | | | | |
| ATTRIBUTES | <p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>Availability</td> <td>SUNWtnfd</td> </tr> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | Availability | SUNWtnfd | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| Availability | SUNWtnfd | | | | | | |
| MT-Level | MT-Safe | | | | | | |
| SEE ALSO | <code>prex(1)</code> , <code>tnfdump(1)</code> , <code>TNF_DECLARE_RECORD(3X)</code> , <code>TNF_PROBE(3X)</code> , <code>attributes(5)</code> | | | | | | |

NOTES

A probe is considered enabled only if:

- **prex(1)** has enabled the probe AND
- the process has probing enabled — which is the default or could be set via **tnf_process_enable()** AND
- the thread that hits the probe has probing enabled — which is every thread's default or could be set via `tnf_thread_enable()`.

There is a run time cost associated with determining that the probe is disabled. To reduce the performance effect of probes, this cost should be minimized. The quickest way that a probe can be determined to be disabled is by the enable control that **prex(1)** uses. Therefore, to disable all the probes in a process use the `disable` command in **prex(1)** rather than `tnf_process_disable()`.

tnf_process_disable() and **tnf_process_enable()** should only be used to toggle probing based on some internal program condition. **tnf_thread_disable()** should be used to turn off probing for threads that are uninteresting.

NAME toascii – translate integer to a 7-bit ASCII character

SYNOPSIS #include <ctype.h>
int toascii(int c);

DESCRIPTION The **toascii()** function converts its argument into a 7-bit ASCII character.

RETURN VALUES The **toascii()** function returns the value (*c* & 0x7E).

ERRORS No errors are returned.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |
| CSI | Enabled |

SEE ALSO **isascii(3C)**, **attributes(5)**

NAME `_tolower` – transliterate upper-case characters to lower-case

SYNOPSIS `#include <ctype.h>`
`int _tolower(int c);`

DESCRIPTION The `_tolower()` macro is equivalent to `tolower(3C)` except that the argument `c` must be an upper-case letter.

RETURN VALUES On successful completion, `_tolower()` returns the lower-case letter corresponding to the argument passed.

ERRORS No errors are defined.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |
| CSI | Enabled |

SEE ALSO `isupper(3C)`, `tolower(3C)`, `attributes(5)`

| NAME | tolower – transliterate upper-case characters to lower-case | | | | | | |
|----------------------|---|----------------|-----------------|----------|---------|-----|---------|
| SYNOPSIS | #include <ctype.h> int tolower(int c); | | | | | | |
| DESCRIPTION | The tolower() function has as a domain a type <code>int</code> , the value of which is representable as an <code>unsigned char</code> or the value of <code>EOF</code> . If the argument has any other value, the argument is returned unchanged. If the argument of tolower() represents an upper-case letter, and there exists a corresponding lower-case letter (as defined by character type information in the program locale category <code>LC_CTYPE</code>), the result is the corresponding lower-case letter. All other arguments in the domain are returned unchanged. | | | | | | |
| RETURN VALUES | On successful completion, tolower() returns the lower-case letter corresponding to the argument passed. Otherwise, it returns the argument unchanged. | | | | | | |
| ERRORS | No errors are defined. | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> <tr> <td>CSI</td> <td>Enabled</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe | CSI | Enabled |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| MT-Level | MT-Safe | | | | | | |
| CSI | Enabled | | | | | | |
| SEE ALSO | _tolower(3C) , setlocale(3C) , attributes(5) | | | | | | |

| | |
|--------------------|---|
| NAME | t_open – establish a transport endpoint |
| SYNOPSIS | <pre>#include <xti.h> #include <fcntl.h></pre> <pre>int t_open(const char *name, int oflag, struct t_info *info);</pre> |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>The <code>t_open()</code> function must be called as the first step in the initialization of a transport endpoint. This function establishes a transport endpoint by supplying a transport provider identifier that indicates a particular transport provider, that is, transport protocol, and returning a file descriptor that identifies that endpoint.</p> <p>The argument <code>name</code> points to a transport provider identifier and <code>oflag</code> identifies any open flags, as in <code>open(2)</code>. The argument <code>oflag</code> is constructed from <code>O_RDWR</code> optionally bitwise inclusive-OR'ed with <code>O_NONBLOCK</code>. These flags are defined by the header <code><fcntl.h></code>. The file descriptor returned by <code>t_open()</code> will be used by all subsequent functions to identify the particular local transport endpoint.</p> <p>This function also returns various default characteristics of the underlying transport protocol by setting fields in the <code>t_info</code> structure. This argument points to a <code>t_info</code> which contains the following members:</p> <pre>t_scalar_t addr; /* max size of the transport protocol address */ t_scalar_t options; /* max number of bytes of */ /* protocol-specific options */ t_scalar_t tsdu; /* max size of a transport service data */ /* unit (TSDU) */ t_scalar_t etsdu; /* max size of an expedited transport */ /* service data unit (ETSDU) */ t_scalar_t connect; /* max amount of data allowed on */ /* connection establishment functions */ t_scalar_t discon; /* max amount of data allowed on */ /* t_snddis() and t_rcvdis() functions */ t_scalar_t servtype; /* service type supported by the */ /* transport provider */ t_scalar_t flags; /* other info about the transport provider */</pre> |

The values of the fields have the following meanings:

- addr** A value greater than zero (T_NULL) indicates the maximum size of a transport protocol address and a value of -2 (T_INVALID) specifies that the transport provider does not provide user access to transport protocol addresses.
- options** A value greater than zero (T_NULL) indicates the maximum number of bytes of protocol-specific options supported by the provider, and a value of -2 (T_INVALID) specifies that the transport provider does not support user-settable options.
- tsdu** A value greater than zero (T_NULL) specifies the maximum size of a transport service data unit (TSDU); a value of zero (T_NULL) specifies that the transport provider does not support the concept of TSDU, although it does support the sending of a data stream with no logical boundaries preserved across a connection; a value of -1 (T_INFINITE) specifies that there is no limit to the size of a TSDU; and a value of -2 (T_INVALID) specifies that the transfer of normal data is not supported by the transport provider.
- etsdu** A value greater than zero (T_NULL) specifies the maximum size of an expedited transport service data unit (ETSDU); a value of zero (T_NULL) specifies that the transport provider does not support the concept of ETSDU, although it does support the sending of an expedited data stream with no logical boundaries preserved across a connection; a value of -1 (T_INFINITE) specifies that there is no limit on the size of an ETSDU; and a value of -2 (T_INVALID) specifies that the transfer of expedited data is not supported by the transport provider. Note that the semantics of expedited data may be quite different for different transport providers.
- connect** A value greater than zero (T_NULL) specifies the maximum amount of data that may be associated with connection establishment functions, and a value of -2 (T_INVALID) specifies that the transport provider does not allow data to be sent with connection establishment functions.
- discon** If the T_ORDRELDATA bit in flags is clear, a value greater than zero (T_NULL) specifies the maximum amount of data that may be associated with the `t_snddis(3N)` and `t_rcvdis(3N)` functions, and a value of -2 (T_INVALID)

specifies that the transport provider does not allow data to be sent with the abortive release functions. If the `T_ORDRELDATA` bit is set in `flags`, a value greater than zero (`T_NULL`) specifies the maximum number of octets that may be associated with the `t_sndreldata()`, `t_rcvreldata()`, `t_snddis(3N)` and `t_rcvdis(3N)` functions.

servtype This field specifies the service type supported by the transport provider, as described below.

flags This is a bit field used to specify other information about the communications provider. If the `T_ORDRELDATA` bit is set, the communications provider supports user data to be sent with an orderly release. If the `T_SENDZERO` bit is set in `flags`, this indicates the underlying transport provider supports the sending of zero-length TSDUs.

If a transport user is concerned with protocol independence, the above sizes may be accessed to determine how large the buffers must be to hold each piece of information. Alternatively, the `t_alloc(3N)` function may be used to allocate these buffers. An error will result if a transport user exceeds the allowed data size on any function.

The *servtype* field of *info* specifies one of the following values on return:

`T_COTS` The transport provider supports a connection-mode service but does not support the optional orderly release facility.

`T_COTS_ORD` The transport provider supports a connection-mode service with the optional orderly release facility.

`T_CLTS` The transport provider supports a connectionless-mode service. For this service type, `t_open()` will return `-2` (`T_INVALID`) for *etsdu*, *connect* and *discon*.

A single transport endpoint may support only one of the above services at one time.

If *info* is set to a null pointer by the transport user, no protocol information is returned by `t_open()`.

RETURN VALUES

A valid file descriptor is returned upon successful completion. Otherwise, a value of `-1` is returned and `t_errno` is set to indicate an error.

VALID STATES

`T_UNINIT.`

ERRORS

On failure, `t_errno` is set to the following:

`TBADFLAG` An invalid flag is specified.

| | | |
|---------------------------------|----------|--|
| | TBADNAME | Invalid transport provider name. |
| | TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). |
| | TSYSERR | A system error has occurred during execution of this function. |
| TLI COMPATIBILITY | | The XTI and TLI interface definitions have common names but use different header files. This and other semantic differences between the two interfaces are described in the subsections below. |
| Interface Header | | The XTI interfaces use the <code>xti.h</code> TLI interfaces should <i>not</i> use this header. They should use the header: <pre>#include <tiuser.h></pre> |
| Error Description Values | | The <code>t_errno</code> values TPROTO and TBADNAME can be set by the XTI interface but cannot be set by the TLI interface. |
| Notes | | For TLI, the <code>t_info</code> structure referenced by <i>info</i> lacks the following structure member: <pre>t_scalar_t flags; /* other info about the transport provider */</pre> This member was added to <code>struct t_info</code> in the XTI interfaces. When a value of -1 is observed as the return value in various <code>t_info</code> structure members, it signifies that the transport provider can handle an infinite length buffer for a corresponding attribute, such as address data, option data, TSDU (octet size), ETSDU (octet size), connection data, and disconnection data. The corresponding structure members are <code>addr</code> , <code>options</code> , <code>tsdu</code> , <code>estdu</code> , <code>connect</code> , and <code>discon</code> , respectively. For more information refer to the <i>Transport Interfaces Programming Guide</i> |
| ATTRIBUTES | | See <code>attributes(5)</code> for descriptions of the following attributes: |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO`open(2)`, `attributes(5)`*Transport Interfaces Programming Guide*

| | |
|--------------------|---|
| NAME | t_optmgmt – manage options for a transport endpoint |
| SYNOPSIS | <pre>#include <xti.h> int t_optmgmt(int fd, const struct t_optmgmt *req, struct t_optmgmt *ret);</pre> |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>The <code>t_optmgmt()</code> function enables a transport user to retrieve, verify or negotiate protocol options with the transport provider. The argument <code>fd</code> identifies a transport endpoint.</p> <p>The <code>req</code> and <code>ret</code> arguments point to a <code>t_optmgmt</code> structure containing the following members:</p> <pre>struct netbuf opt; t_scalar_t flags;</pre> <p>The <code>opt</code> field identifies protocol options and the <code>flags</code> field is used to specify the action to take with those options.</p> <p>The options are represented by a <code>netbuf</code> structure in a manner similar to the address in <code>t_bind(3N)</code>. The argument <code>req</code> is used to request a specific action of the provider and to send options to the provider. The argument <code>len</code> specifies the number of bytes in the options, <code>buf</code> points to the options buffer, and <code>maxlen</code> has no meaning for the <code>req</code> argument. The transport provider may return options and flag values to the user through <code>ret</code>. For <code>ret</code>, <code>maxlen</code> specifies the maximum size of the options buffer and <code>buf</code> points to the buffer where the options are to be placed. If <code>maxlen</code> in <code>ret</code> is set to zero, no options values are returned. On return, <code>len</code> specifies the number of bytes of options returned. The value in <code>maxlen</code> has no meaning for the <code>req</code> argument, but must be set in the <code>ret</code> argument to specify the maximum number of bytes the options buffer can hold.</p> <p>Each option in the options buffer is of the form <code>struct t_opthdr</code> possibly followed by an option value.</p> <p>The <code>level</code> field of <code>struct t_opthdr</code> identifies the XTI level or a protocol of the transport provider. The <code>name</code> field identifies the option within the level, and <code>len</code> contains its total length; that is, the length of the option header <code>t_opthdr</code> plus the length of the option value. If <code>t_optmgmt()</code> is called with</p> |

the action `T_NEGOTIATE` set, the *status* field of the returned options contains information about the success or failure of a negotiation.

Several options can be concatenated. The option user has, however to ensure that each options header and value part starts at a boundary appropriate for the architecture-specific alignment rules. The macros `T_OPT_FIRSTHDR(nbp)`, `T_OPT_NEXTHDR (nbp,tohp)`, `T_OPT_DATA(tohp)` are provided for that purpose.

`T_OPT_DATA(nhp)`

If argument is a pointer to a `t_opthdr` structure, this macro returns an unsigned character pointer to the data associated with the `t_opthdr`.

`T_OPT_NEXTHDR(nbp, tohp)`

If the first argument is a pointer to a `netbuf` structure associated with an option buffer and second argument is a pointer to a `t_opthdr` structure within that option buffer, this macro returns a pointer to the next `t_opthdr` structure or a null pointer if this `t_opthdr` is the last `t_opthdr` in the option buffer.

`T_OPT_FIRSTHDR(tohp)`

If the argument is a pointer to a `netbuf` structure associated with an option buffer, this macro returns the pointer to the first `t_opthdr` structure in the associated option buffer, or a null pointer if there is no option buffer associated with this `netbuf` or if it is not possible or the associated option buffer is too small to accommodate even the first aligned option header.

`T_OPT_FIRSTHDR` is useful for finding an appropriately aligned start of the option buffer. `T_OPT_NEXTHDR` is useful for moving to the start of the next appropriately aligned option in the option buffer. Note that `OPT_NEXTHDR` is also available for backward compatibility requirements. `T_OPT_DATA` is useful for finding the start of the data part in the option

buffer where the contents of its values start on an appropriately aligned boundary.

If the transport user specifies several options on input, all options must address the same level.

If any option in the options buffer does not indicate the same level as the first option, or the level specified is unsupported, then the **t_optmgmt()** request will fail with `TBADOPT`. If the error is detected, some options have possibly been successfully negotiated. The transport user can check the current status by calling **t_optmgmt()** with the `T_CURRENT` flag set.

The *flags* field of *req* must specify one of the following actions:

`T_NEGOTIATE`

This action enables the transport user to negotiate option values.

The user specifies the options of interest and their values in the buffer specified by *req*→*opt.buf* and *req*→*opt.len*. The negotiated option values are returned in the buffer pointed to by *ret*→*opt.buf*. The *status* field of each returned option is set to indicate the result of the negotiation. The value is `T_SUCCESS` if the proposed value was negotiated, `T_PARTSUCCESS` if a degraded value was negotiated, `T_FAILURE` if the negotiation failed (according to the negotiation rules), `T_NOTSUPPORT` if the transport provider does not support this option or illegally requests negotiation of a privileged option, and `T_READONLY` if modification of a read-only option was requested. If the status is `T_SUCCESS`, `T_FAILURE`, `T_NOTSUPPORT` or `T_READONLY`, the

returned option value is the same as the one requested on input.

The overall result of the negotiation is returned in *ret→flags*.

This field contains the worst single result, whereby the rating is done according to the order

T_NOTSUPPORT, T_READONLY,
T_FAILURE, T_PARTSUCCESS,
T_SUCCESS. The value

T_NOTSUPPORT is the worst result and T_SUCCESS is the best.

For each level, the option T_ALLOPT can be requested on input. No value is given with this option; only the *t_opthdr* part is specified. This input requests to negotiate all supported options of this level to their default values. The result is returned option by option in *ret→opt.buf*. Note that depending on the state of the transport endpoint, not all requests to negotiate the default value may be successful.

T_CHECK

This action enables the user to verify whether the options specified in *req* are supported by the transport provider. If an option is specified with no option value (it consists only of a *t_opthdr* structure), the option is returned with its *status* field set to T_SUCCESS if it is supported, T_NOTSUPPORT if it is not or needs additional user privileges, and T_READONLY if it is read-only (in the current XTI state). No option value is returned.

If an option is specified with an option value, the *status* field of the returned option has the same value, as if the user had tried to negotiate this value with T_NEGOTIATE. If the

status is T_SUCCESS, T_FAILURE, T_NOTSUPPORT or T_READONLY, the returned option value is the same as the one requested on input.

The overall result of the option checks is returned in *ret→flags*. This field contains the worst single result of the option checks, whereby the rating is the same as for T_NEGOTIATE .

Note that no negotiation takes place. All currently effective option values remain unchanged.

T_DEFAULT

This action enables the transport user to retrieve the default option values. The user specifies the options of interest in *req→opt.buf*. The option values are irrelevant and will be ignored; it is sufficient to specify the *t_opthdr* part of an option only. The default values are then returned in *ret→opt.buf*.

The *status* field returned is T_NOTSUPPORT if the protocol level does not support this option or the transport user illegally requested a privileged option, T_READONLY if the option is read-only, and set to T_SUCCESS in all other cases. The overall result of the request is returned in *ret→flags*. This field contains the worst single result, whereby the rating is the same as for T_NEGOTIATE .

For each level, the option T_ALLOPT can be requested on input. All supported options of this level with their default values are then returned. In this case, *ret→opt.maxlen* must be given at least the value *info→options* before the call. See *t_getinfo*(3N) and *t_open*(3N).

T_CURRENT

This action enables the transport user to retrieve the currently effective option values. The user specifies the options of interest in *req→opt.buf*. The option values are irrelevant and will be ignored; it is sufficient to specify the *t_opthdr* part of an option only. The currently effective values are then returned in *req→opt.buf*.

The *status* field returned is T_NOTSUPPORT if the protocol level does not support this option or the transport user illegally requested a privileged option, T_READONLY if the option is read-only, and set to T_SUCCESS in all other cases. The overall result of the request is returned in *ret→flags*. This field contains the worst single result, whereby the rating is the same as for T_NEGOTIATE.

For each level, the option T_ALLOPT can be requested on input. All supported options of this level with their currently effective values are then returned.

The option T_ALLOPT can only be used with **t_optmgmt()** and the actions T_NEGOTIATE, T_DEFAULT and T_CURRENT. It can be used with any supported level and addresses all supported options of this level. The option has no value; it consists of a *t_opthdr* only. Since in a **t_optmgmt()** call only options of one level may be addressed, this option should not be requested together with other options. The function returns as soon as this option has been processed.

Options are independently processed in the order they appear in the input option buffer. If an option is multiply

input, it depends on the implementation whether it is multiply output or whether it is returned only once.

Transport providers may not be able to provide an interface capable of supporting T_NEGOTIATE and/or T_CHECK functionalities. When this is the case, the error TNOTSUPPORT is returned.

The function **t_optmgmt()** may block under various circumstances and depending on the implementation. The function will block, for instance, if the protocol addressed by the call resides on a separate controller. It may also block due to flow control constraints; that is, if data sent previously across this transport endpoint has not yet been fully processed. If the function is interrupted by a signal, the option negotiations that have been done so far may remain valid. The behavior of the function is not changed if O_NONBLOCK is set.

RETURN VALUES

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and `t_errno` is set to indicate an error.

VALID STATES

ALL - apart from T_UNINIT.

ERRORS

On failure, `t_errno` is set to one of the following:

| | |
|-----------|--|
| TBADF | The specified file descriptor does not refer to a transport endpoint. |
| TBADFLAG | An invalid flag was specified. |
| TBADOPT | The specified options were in an incorrect format or contained illegal information. |
| TBUFOVFLW | The number of bytes allowed for an incoming argument (<i>maxlen</i>) is greater than 0 but not sufficient to store the value |

| | |
|---------------------------------|--|
| | of that argument. The information to be returned in <i>ret</i> will be discarded. |
| | TNOTSUPPORT This action is not supported by the transport provider. |
| | TOUTSTATE The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. |
| | TPROTO This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<i>t_errno</i>). |
| | TSYSERR A system error has occurred during execution of this function. |
| TLI COMPATIBILITY | The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below. |
| Interface Header | The XTI interfaces use the header file, <i>xti.h</i> . TLI interfaces should <i>not</i> use this header. They should use the header: <pre>#include <tiuser.h></pre> |
| Error Description Values | The <i>t_errno</i> value TPROTO can be set by the XTI interface but not by the TLI interface. The <i>t_errno</i> values that this routine can return under different circumstances than its XTI counterpart are TACCES and TBUFOVFLW. TACCES can be returned to indicate that the user does not have permission to negotiate the specified options. TBUFOVFLW can be returned even when the <i>maxlen</i> field of the corresponding buffer has been set to zero. |
| Option Buffers | The format of the options in an <i>opt</i> buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not fix the buffer format. The macros T_OPT_DATA, T_OPT_NEXTHDR, and T_OPT_FIRSTHDR described for XTI are not available for use by TLI interfaces. |
| Actions | The semantic meaning of various action values for the <i>flags</i> field of <i>req</i> differs between the TLI and XTI interfaces. TLI interface users should heed the following descriptions of the actions: T_NEGOTIATE This action enables the user to negotiate the values of the options specified in <i>req</i> with the transport provider. The |

provider will evaluate the requested options and negotiate the values, returning the negotiated values through *ret*.

T_CHECK This action enables the user to verify whether the options specified in *req* are supported by the transport provider. On return, the *flags* field of *ret* will have either **T_SUCCESS** or **T_FAILURE** set to indicate to the user whether the options are supported. These flags are only meaningful for the **T_CHECK** request.

T_DEFAULT This action enables a user to retrieve the default options supported by the transport provider into the *opt* field of *ret*. In *req*, the *len* field of *opt* must be zero and the *buf* field may be **NULL**.

Connectionless-Mode

If issued as part of the connectionless-mode service, **t_optmgmt()** may block due to flow control constraints. The function will not complete until the transport provider has processed all previously sent data units.

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

close(2), **poll(2)**, **select(3C)**, **t_accept(3N)**, **t_alloc(3N)**, **t_bind(3N)**, **t_close(3N)**, **t_connect(3N)**, **t_getinfo(3N)**, **t_listen(3N)**, **t_open(3N)**, **t_rcv(3N)**, **t_rcvconnect(3N)**, **t_rcvudata(3N)**, **t_snddis(3N)**, **attributes(5)**

Transport Interfaces Programming Guide

NAME | _toupper – transliterate lower-case characters to upper-case

SYNOPSIS | #include <ctype.h>
| int _toupper(int c);

DESCRIPTION | The **_toupper()** macro is equivalent to **toupper(3C)** except that the argument *c* must be a lower-case letter.

RETURN VALUES | On successful completion, **_toupper()** returns the upper-case letter corresponding to the argument passed.

ERRORS | No errors are defined.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |
| CSI | Enabled |

SEE ALSO | **islower(3C)**, **toupper(3C)**, **attributes(5)**

| NAME | toupper – transliterate lower-case characters to upper-case | | | | | | |
|----------------------|--|----------------|-----------------|----------|---------|-----|---------|
| SYNOPSIS | #include <ctype.h> int toupper (int c); | | | | | | |
| DESCRIPTION | The toupper() function has as a domain a type <code>int</code> , the value of which is representable as an <code>unsigned char</code> or the value of <code>EOF</code> . If the argument has any other value, the argument is returned unchanged. If the argument of toupper() represents a lower-case letter, and there exists a corresponding upper-case letter (as defined by character type information in the program locale category <code>LC_CTYPE</code>), the result is the corresponding upper-case letter. All other arguments in the domain are returned unchanged. | | | | | | |
| RETURN VALUES | On successful completion, toupper() returns the upper-case letter corresponding to the argument passed. | | | | | | |
| ERRORS | No errors are defined. | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> <tr> <td>CSI</td> <td>Enabled</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe | CSI | Enabled |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| MT-Level | MT-Safe | | | | | | |
| CSI | Enabled | | | | | | |
| SEE ALSO | _toupper(3C) , setlocale(3C) , attributes(5) | | | | | | |

NAME towctrans – wide-character mapping

SYNOPSIS #include <wctype.h>

```
wint_t towctrans(wint_t wc, wctrans_t desc);
```

DESCRIPTION

The **towctrans()** function maps the wide character *wc* using the mapping described by *desc*. The current setting of the LC_CTYPE category shall be the same as during the call to **wctrans()** that returned the value *desc*.

The function call `towctrans(wc, wctrans("tolower"))` behaves the same as `towlower(wc)`.

The function call `towctrans(wc, wctrans("toupper"))` behaves the same as `toupper(wc)`.

RETURN VALUES

The **towctrans()** function returns the mapped value of *wc*, using the mapping described by *desc*; otherwise, it returns *wc* unchanged.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |
| CSI | Enabled |

SEE ALSO

setlocale(3C), **wctrans(3C)**, **attributes(5)**

NAME towlower – transliterate upper-case wide-character code to lower-case

SYNOPSIS `#include <wchar.h>`
`wint_t towlower(wint_t wc);`

DESCRIPTION The **towlower()** function has as a domain a type `wint_t`, the value of which must be a character representable as a `wchar_t`, and must be a wide-character code corresponding to a valid character in the current locale or the value of `WEOF`. If the argument has any other value, the argument is returned unchanged. If the argument of **towlower()** represents an upper-case wide-character code, and there exists a corresponding lower-case wide-character code (as defined by character type information in the program locale category `LC_CTYPE`), the result is the corresponding lower-case wide-character code. All other arguments in the domain are returned unchanged.

RETURN VALUES On successful completion, **towlower()** returns the lower-case letter corresponding to the argument passed. Otherwise, it returns the argument unchanged.

ERRORS No errors are defined.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |
| CSI | Enabled |

SEE ALSO `iswalpha(3C)`, `setlocale(3C)`, `towupper(3C)`, **attributes(5)**

NAME | towupper – transliterate lower-case wide-character code to upper-case

SYNOPSIS | #include <wchar.h>

| wint_t towupper(wint_t wc);

DESCRIPTION | The **towupper()** function has as a domain a type `wint_t`, the value of which must be a character representable as a `wchar_t`, and must be a wide-character code corresponding to a valid character in the current locale or the value of `WEOF`. If the argument has any other value, the argument is returned unchanged. If the argument of **towupper()** represents a lower-case wide-character code (as defined by character type information in the program locale category `LC_CTYPE`), the result is the corresponding upper-case wide-character code. All other arguments in the domain are returned unchanged.

RETURN VALUES | Upon successful completion, **towupper()** returns the upper-case letter corresponding to the argument passed. Otherwise, it returns the argument unchanged.

ERRORS | No errors are defined.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |
| CSI | Enabled |

SEE ALSO | **iswalph(3C)**, **setlocale(3C)**, **towlower(3C)**, **attributes(5)**

| NAME | tracing - overview of tnf tracing system |
|--------------------|--|
| DESCRIPTION | <p>tnf tracing is a set of programs and API's that can be used to present a high-level view of the performance of an executable, a library, or part of the kernel. tracing is used to analyze a program's performance and identify the conditions that produced a bug.</p> <p>The core elements of tracing are:</p> |
| | <p>TNF_PROBE_*() The TNF_PROBE_*() macros define "probes" to be placed in code which, when enabled and executed, cause information to be added to a trace file. See TNF_PROBE(3X). If there are insufficient TNF_PROBE_* macros to store all the data of interest for a probe, data may be grouped into records. See TNF_DECLARE_RECORD(3X).</p> |
| | <p>prex Displays and controls probes in running software. See prex(1).</p> |
| | <p>kernel probes A set of probes built into the Solaris kernel which capture information about system calls, multithreading, page faults, swapping, memory management, and I/O. You can use these probes to obtain detailed traces of kernel activity under your application workloads. See tnf_kernel_probes(4).</p> |
| | <p>tnfextract A program that extracts the trace data from the kernel's in-memory buffer into a file. See tnfextract(1).</p> |
| | <p>tnfdump A program that displays the information from a trace file. See tnfdump(1).</p> |
| | <p>libtnfctl A library of interfaces that controls probes in a process. See libtnfctl(3X). prex(1) also utilizes this library. Other tools and processes use the libtnfctl interfaces to exercise fine control over their own probes.</p> |
| | <p>tnf_process_enable() A routine called by a process to turn on tracing and probe functions for the current process. See tnf_process_enable(3X).</p> |

| | |
|------------------------------|---|
| tnf_process_disable() | A routine called by a process to turn off tracing and probe functions for the current process. See tnf_process_disable(3X) . |
| tnf_thread_enable() | A routine called by a process to turn on tracing and probe functions for the currently running thread. See tnf_thread_enable(3X) . |
| tnf_thread_disable() | A routine called by a process to turn off tracing and probe functions for the currently running thread. See tnf_thread_disable(3X) . |

EXAMPLES**EXAMPLE 1** Tracing.

The two examples shown here illustrate tracing within a process and within the kernel.

Tracing a Process

The following function in some daemon process accepts job requests of various types, queueing them for later execution. There are two "debug probes" and one "production probe." Note that probes which are intended for debugging will not be compiled into the final version of the code; however, production probes are compiled into the final product.

```

/*
 * To compile in all probes (for development):
 * cc -DTNF_DEBUG ...
 *
 * To compile in only production probes (for release):
 * cc ...
 *
 * To compile in no probes at all:
 * cc -DNPROBE ...
 */
#include <tnf/probe.h>
void work(long, char *);
enum work_request_type { READ, WRITE, ERASE, UPDATE };
static char *work_request_name[] = {"read", "write", "erase", "update"};
main()
{
    long i;
    for (i = READ; i <= UPDATE; i++)
        work(i, work_request_name[i]);
}
void work(long request_type, char *request_name)
{
    static long q_length;
    TNF_PROBE_2_DEBUG(work_start, "work",
        "XYZ%debug 'in function work'",
        tnf_long, request_type_arg, request_type,
        tnf_string, request_name_arg, request_name);
    /* assume work request is queued for later processing */
    q_length++;
}

```

```

    TNF_PROBE_1(work_queue, "work queue",
"XYZ%work_load heavy",
tnf_long, queue_length, q_length);
    TNF_PROBE_0_DEBUG(work_end, "work", "");
}

```

The production probe "work_queue," which remains compiled in the code, will, when enabled, log the length of the work queue each time a request is received.

The debug probes "work_start" and "work_end, " which are compiled only during the development phase, track entry to and exit from the `work()` function and measure how much time is spent executing it. Additionally, the debug probe "work_start" logs the value of the two incoming arguments `request_type` and `request_name`. The runtime overhead for disabled probes is low enough that one can liberally embed them in the code with little impact on performance.

For debugging, the developer would compile with `-DTNF_DEBUG`, run the program under control of `prex(1)`, enable the probes of interest (in this case, all probes), continue the program until exit, and dump the trace file:

```

% cc
-DTNF_DEBUG -o daemon daemon.c # compile in all probes
% prex daemon # run program under prex control
Target process stopped
Type "continue" to resume the target, "help" for help ...
prex> list probes $all # list all probes in program
<probe list output here>
prex> enable $all # enable all probes
prex> continue # let target process execute
<program output here>
prex: target process finished
% ls /tmp/trace-* # trace output is in trace-<pid>
/tmp/trace-4194
% tnfdump /tmp/trace-4194 # get ascii output of trace file
<trace records output here>

```

For the production version of the system, the developer simply compiles without `--DTNF_DEBUG`.

Tracing the Kernel

Kernel tracing is similar to tracing a process; however, there are some differences. For instance, to trace the kernel, you need superuser privileges. The following example uses `prex(1)` and traces the probes in the kernel that capture system call information.

```

Allocate kernel
trace buffer and capture trace data:
root# prex -k

```

```

Type "help" for help ...
prex> buffer alloc 2m    # allocate kernel trace buffer
Buffer of size 2097152 bytes allocated
prex> list probes $all   # list all kernel probes
<probe list output here>
prex> list probes syscall # list syscall probes
                        # (keys=syscall)
<syscall probes list output here>
prex> enable syscall    # enable only syscall probes
prex> ktrace on         # turn on kernel tracing
<Run your application in another window at this point>
prex> ktrace off        # turn off kernel tracing
prex> quit              # exit prex
Extract the kernel's trace buffer into a file:
root# tnfxtract /tmp/ktrace # extract kernel trace buffer
Reset kernel tracing:
root# prex -k
prex> disable $all      # disable all probes
prex> untrace $all      # untrace all probes
prex> buffer dealloc    # deallocate kernel trace buffer
prex> quit

```

CAUTION: Do not deallocate the trace buffer until you have extracted it into a trace file. Otherwise, you will lose the trace data that you collected from your experiment!

Examine the kernel trace file:

```

root# tnfdump /tmp/ktrace # get ascii dump of trace file
<trace records output here>

```

prex can also attach to a running process, list probes, and perform a variety of other tasks. For more detailed examples and a more thorough discussion of tracing under Solaris, see the chapter entitled "Tracing Program Execution with the TNF Utilities" in the *Programming Utilities Guide*

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| Availability | SUNWtnfd |
| MT Level | MT-Safe |

SEE ALSO

prex(1), **tnfdump(1)**, **tnfxtract(1)**, **TNF_DECLARE_RECORD(3X)**, **TNF_PROBE(3X)**, **libtnfctl(3X)**, **tnf_process_disable(3X)**, **tnf_kernel_probes(4)**, **attributes(5)**

Programming Utilities Guide

| | |
|--------------------|--|
| NAME | t_rcv – receive data or expedited data sent over a connection |
| SYNOPSIS | <pre>#include <xti.h> int t_rcv(int fd, void *buf, unsigned int nbytes, int *flags);</pre> |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the <code>TLI COMPATIBILITY</code> section for a description of differences between the two interfaces.</p> <p>This function receives either normal or expedited data. The argument <i>fd</i> identifies the local transport endpoint through which data will arrive, <i>buf</i> points to a receive buffer where user data will be placed, and <i>nbytes</i> specifies the size of the receive buffer. The argument <i>flags</i> may be set on return from <code>t_rcv()</code> and specifies optional flags as described below.</p> <p>By default, <code>t_rcv()</code> operates in synchronous mode and will wait for data to arrive if none is currently available. However, if <code>O_NONBLOCK</code> is set by means of <code>t_open(3N)</code> or <code>fcntl(2)</code>, <code>t_rcv()</code> will execute in asynchronous mode and will fail if no data is available. See <code>TNODATA</code> below.</p> <p>On return from the call, if <code>T_MORE</code> is set in <i>flags</i>, this indicates that there is more data, and the current transport service data unit (TSDU) or expedited transport service data unit (ETSDU) must be received in multiple <code>t_rcv()</code> calls. In the asynchronous mode, or under unusual conditions (for example, the arrival of a signal or <code>T_EXDATA</code> event), the <code>T_MORE</code> flag may be set on return from the <code>t_rcv()</code> call even when the number of bytes received is less than the size of the receive buffer specified. Each <code>t_rcv()</code> with the <code>T_MORE</code> flag set indicates that another <code>t_rcv()</code> must follow to get more data for the current TSDU. The end of the TSDU is identified by the return of a <code>t_rcv()</code> call with the <code>T_MORE</code> flag not set. If the transport provider does not support the concept of a TSDU as indicated in the <i>info</i> argument on return from <code>t_open(3N)</code> or <code>t_getinfo(3N)</code>, the <code>T_MORE</code> flag is not meaningful and should be ignored. If <i>nbytes</i> is greater than zero on the call to <code>t_rcv()</code>, <code>t_rcv()</code> will return 0 only if the end of a TSDU is being returned to the user.</p> <p>On return, the data is expedited if <code>T_EXPEDITED</code> is set in <i>flags</i>. If <code>T_MORE</code> is also set, it indicates that the number of expedited bytes exceeded <i>nbytes</i>, a signal has interrupted the call, or that an entire ETSDU was not available (only for transport protocols that support fragmentation of ETSDUs). The rest of the ETSDU will be returned by subsequent calls to <code>t_rcv()</code> which will return with <code>T_EXPEDITED</code> set in <i>flags</i>. The end of the ETSDU is identified by the return of a <code>t_rcv()</code> call with <code>T_EXPEDITED</code> set and <code>T_MORE</code> cleared. If the entire ETSDU</p> |

is not available it is possible for normal data fragments to be returned between the initial and final fragments of an ETSDU.

If a signal arrives, **t_rcv()** returns, giving the user any data currently available. If no data is available, **t_rcv()** returns -1, sets **t_errno** to **TSYSERR** and **errno** to **EINTR**. If some data is available, **t_rcv()** returns the number of bytes received and **T_MORE** is set in flags.

In synchronous mode, the only way for the user to be notified of the arrival of normal or expedited data is to issue this function or check for the **T_DATA** or **T_EXDATA** events using the **t_look(3N)** function. Additionally, the process can arrange to be notified by means of the EM interface.

RETURN VALUES

On successful completion, **t_rcv()** returns the number of bytes received. Otherwise, it returns 1 on failure and **t_errno** is set to indicate the error.

VALID STATES

T_DATAXFER, **T_OUTREL**.

ERRORS

On failure, **t_errno** is set to one of the following:

| | |
|--------------------|---|
| TBADF | The specified file descriptor does not refer to a transport endpoint. |
| TLOOK | An asynchronous event has occurred on this transport endpoint and requires immediate attention. |
| TNODATA | O_NONBLOCK was set, but no data is currently available from the transport provider. |
| TNOTSUPPORT | This function is not supported by the underlying transport provider. |
| TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (t_errno). |
| TSYSERR | A system error has occurred during execution of this function. |

TLI COMPATIBILITY

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, **xti.h**. TLI interfaces should *not* use this header. They should use the header:

```
#include <tiuser.h>
```

**Error Description
Values**

The `t_errno` value that can be set by the XTI interface and cannot be set by the TLI interface is:

TPROTO

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`fcntl(2)`, `t_getinfo(3N)`, `t_look(3N)`, `t_open(3N)`, `t_snd(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

| | |
|--------------------|---|
| NAME | t_rcvconnect – receive the confirmation from a connection request |
| SYNOPSIS | <pre>#include <xti.h> int t_rcvconnect(int fd, struct t_call *call);</pre> |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>This function enables a calling transport user to determine the status of a previously sent connection request and is used in conjunction with <code>t_connect(3N)</code> to establish a connection in asynchronous mode, and to complete a synchronous <code>t_connect(3N)</code> call that was interrupted by a signal. The connection will be established on successful completion of this function.</p> <p>The argument <code>fd</code> identifies the local transport endpoint where communication will be established, and <code>call</code> contains information associated with the newly established connection. The argument <code>call</code> points to a <code>t_call</code> structure which contains the following members:</p> <pre>struct netbuf addr; struct netbuf opt; struct netbuf udata; int sequence;</pre> <p>In <code>call</code>, <code>addr</code> returns the protocol address associated with the responding transport endpoint, <code>opt</code> presents any options associated with the connection, <code>udata</code> points to optional user data that may be returned by the destination transport user during connection establishment, and <code>sequence</code> has no meaning for this function.</p> <p>The <code>maxlen</code> field of each argument must be set before issuing this function to indicate the maximum size of the buffer for each. However, <code>maxlen</code> can be set to zero, in which case no information to this specific argument is given to the user on the return from <code>t_rcvconnect()</code>. If <code>call</code> is set to <code>NULL</code>, no information at all is returned. By default, <code>t_rcvconnect()</code> executes in synchronous mode and waits for the connection to be established before returning. On return, the <code>addr</code>, <code>opt</code> and <code>udata</code> fields reflect values associated with the connection.</p> <p>If <code>O_NONBLOCK</code> is set by means of <code>t_open(3N)</code> or <code>fcntl(2)</code>, <code>t_rcvconnect()</code> executes in asynchronous mode, and reduces to a poll for existing connection confirmations. If none are available, <code>t_rcvconnect()</code> fails and returns</p> |

immediately without waiting for the connection to be established. See `TNODATA` below. In this case, `t_rcvconnect()` must be called again to complete the connection establishment phase and retrieve the information returned in *call*.

RETURN VALUES

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and `t_errno` is set to indicate an error.

VALID STATES

`T_OUTCON`.

ERRORS

On failure, `t_errno` is set to one of the following:

| | |
|--------------------------|--|
| <code>TBADF</code> | The specified file descriptor does not refer to a transport endpoint. |
| <code>TBUFOVFLW</code> | The number of bytes allocated for an incoming argument (<i>maxlen</i>) is greater than 0 but not sufficient to store the value of that argument, and the connection information to be returned in <i>call</i> will be discarded. The provider's state, as seen by the user, will be changed to <code>T_DATAXFER</code> . |
| <code>TLOOK</code> | An asynchronous event has occurred on this transport connection and requires immediate attention. |
| <code>TNODATA</code> | <code>O_NONBLOCK</code> was set, but a connection confirmation has not yet arrived. |
| <code>TNOTSUPPORT</code> | This function is not supported by the underlying transport provider. |
| <code>TOUTSTATE</code> | The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. |
| <code>TPROTO</code> | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). |
| <code>TSYSERR</code> | A system error has occurred during execution of this function. |

**TLI
COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```
#include<tiuser.h>
```

**Error Description
Values**

The `t_errno` value `TPROTO` can be set by the XTI interface but not by the TLI interface.

A `t_errno` value that this routine can return under different circumstances than its XTI counterpart is `TBUFOVFLW`. It can be returned even when the `maxlen` field of the corresponding buffer has been set to zero.

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`fcntl(2)`, `t_accept(3N)`, `t_alloc(3N)`, `t_bind(3N)`, `t_connect(3N)`, `t_listen(3N)`, `t_open(3N)`, `t_optmgmt(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

| | |
|----------------------|---|
| NAME | t_rcvdis – retrieve information from disconnection |
| SYNOPSIS | <pre>#include <xti.h> int t_rcvdis(int fd, struct t_discon *discon);</pre> |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>This function is used to identify the cause of a disconnection and to retrieve any user data sent with the disconnection. The argument <code>fd</code> identifies the local transport endpoint where the connection existed, and <code>discon</code> points to a <code>t_discon</code> structure containing the following members:</p> <pre>struct netbuf udata; int reason; int sequence;</pre> <p>The field <code>reason</code> specifies the reason for the disconnection through a protocol-dependent reason code, <code>udata</code> identifies any user data that was sent with the disconnection, and <code>sequence</code> may identify an outstanding connection indication with which the disconnection is associated. The field <code>sequence</code> is only meaningful when <code>t_rcvdis()</code> is issued by a passive transport user who has executed one or more <code>t_listen(3N)</code> functions and is processing the resulting connection indications. If a disconnection indication occurs, <code>sequence</code> can be used to identify which of the outstanding connection indications is associated with the disconnection.</p> <p>The <code>maxlen</code> field of <code>udata</code> may be set to zero, if the user does not care about incoming data. If, in addition, the user does not need to know the value of <code>reason</code> or <code>sequence</code>, <code>discon</code> may be set to <code>NULL</code> and any user data associated with the disconnection indication shall be discarded. However, if a user has retrieved more than one outstanding connection indication by means of <code>t_listen(3N)</code>, and <code>discon</code> is a null pointer, the user will be unable to identify with which connection indication the disconnection is associated.</p> |
| RETURN VALUES | Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and <code>t_errno</code> is set to indicate an error. |
| VALID STATES | T_DATAXFER, T_OUTCON, T_OUTREL, T_INREL, T_INCON(ocnt > 0). |

ERRORS

On failure, `t_errno` is set to one of the following:

| | |
|-------------|---|
| TBADF | The specified file descriptor does not refer to a transport endpoint. |
| TBUFOVFLW | The number of bytes allocated for incoming data (<i>maxlen</i>) is greater than 0 but not sufficient to store the data. If <i>fd</i> is a passive endpoint with <i>ocnt</i> > 1, it remains in state <code>T_INCON</code> ; otherwise, the endpoint state is set to <code>T_IDLE</code> . |
| TNODIS | No disconnection indication currently exists on the specified transport endpoint. |
| TNOTSUPPORT | This function is not supported by the underlying transport provider. |
| TOUTSTATE | The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. |
| TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). |
| TSYSERR | A system error has occurred during execution of this function. |

**TLI
COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```
#include <tiuser.h>
```

**Error Description
Values**

The `t_errno` values `TPROTO` and `TOUTSTATE` can be set by the XTI interface but not by the TLI interface.

A failure return, and a `t_errno` value that this routine can set under different circumstances than its XTI counterpart is `TBUFOVFLW`. It can be returned even when the `maxlen` field of the corresponding buffer has been set to zero.

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

t_alloc(3N), t_connect(3N), t_listen(3N), t_open(3N),
t_snddis(3N), attributes(5)

Transport Interfaces Programming Guide

| | | | | | | | | | | | |
|--------------------------|--|--------------------|---|--------------------|---|---------------------|---|--------------------------|--|------------------------|--|
| NAME | t_rcvrel – acknowledge receipt of an orderly release indication | | | | | | | | | | |
| SYNOPSIS | <pre>#include <xti.h> int t_rcvrel(int fd);</pre> | | | | | | | | | | |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>This function is used to receive an orderly release indication for the incoming direction of data transfer. The argument <i>fd</i> identifies the local transport endpoint where the connection exists. After receipt of this indication, the user may not attempt to receive more data by means of <code>t_rcv(3N)</code> or <code>t_rcvv()</code>. Such an attempt will fail with <i>t_error</i> set to <code>TOUTSTATE</code>. However, the user may continue to send data over the connection if <code>t_sndrel(3N)</code> has not been called by the user. This function is an optional service of the transport provider, and is only supported if the transport provider returned service type <code>T_COTS_ORD</code> on <code>t_open(3N)</code> or <code>t_getinfo(3N)</code>. Any user data that may be associated with the orderly release indication is discarded when <code>t_rcvrel()</code> is called.</p> | | | | | | | | | | |
| RETURN VALUES | Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and <i>t_errno</i> is set to indicate an error. | | | | | | | | | | |
| VALID STATES | <code>T_DATAXFER</code> , <code>T_OUTREL</code> . | | | | | | | | | | |
| ERRORS | <p>On failure, <i>t_errno</i> is set to one of the following:</p> <table border="0"> <tr> <td style="padding-right: 20px;"><code>TBADF</code></td> <td>The specified file descriptor does not refer to a transport endpoint.</td> </tr> <tr> <td style="padding-right: 20px;"><code>TLOOK</code></td> <td>An asynchronous event has occurred on this transport endpoint and requires immediate attention.</td> </tr> <tr> <td style="padding-right: 20px;"><code>TNOREL</code></td> <td>No orderly release indication currently exists on the specified transport endpoint.</td> </tr> <tr> <td style="padding-right: 20px;"><code>TNOTSUPPORT</code></td> <td>This function is not supported by the underlying transport provider.</td> </tr> <tr> <td style="padding-right: 20px;"><code>TOUTSTATE</code></td> <td>The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid.</td> </tr> </table> | <code>TBADF</code> | The specified file descriptor does not refer to a transport endpoint. | <code>TLOOK</code> | An asynchronous event has occurred on this transport endpoint and requires immediate attention. | <code>TNOREL</code> | No orderly release indication currently exists on the specified transport endpoint. | <code>TNOTSUPPORT</code> | This function is not supported by the underlying transport provider. | <code>TOUTSTATE</code> | The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. |
| <code>TBADF</code> | The specified file descriptor does not refer to a transport endpoint. | | | | | | | | | | |
| <code>TLOOK</code> | An asynchronous event has occurred on this transport endpoint and requires immediate attention. | | | | | | | | | | |
| <code>TNOREL</code> | No orderly release indication currently exists on the specified transport endpoint. | | | | | | | | | | |
| <code>TNOTSUPPORT</code> | This function is not supported by the underlying transport provider. | | | | | | | | | | |
| <code>TOUTSTATE</code> | The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. | | | | | | | | | | |

TPROTO This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (`t_errno`).

TSYSERR A system error has occurred during execution of this function.

TLI COMPATIBILITY

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```
#include<tiuser.h>
```

Error Description Values

The `t_errno` values that can be set by the XTI interface and cannot be set by the TLI interface are:

TPROTO

TOUTSTATE

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`t_getinfo(3N)`, `t_open(3N)`, `t_sndrel(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

| | |
|----------------------|---|
| NAME | t_rcvreldata – receive an orderly release indication or confirmation containing user data |
| SYNOPSIS | <pre>#include <xti.h> int t_rcvreldata(int fd, struct t_discon *discon);</pre> |
| DESCRIPTION | <p>This function is used to receive an orderly release indication for the incoming direction of data transfer and to retrieve any user data sent with the release. The argument <i>fd</i> identifies the local transport endpoint where the connection exists, and <i>discon</i> points to a <code>t_discon</code> structure containing the following members:</p> <pre>struct netbuf udata; int reason; int sequence;</pre> <p>After receipt of this indication, the user may not attempt to receive more data by means of <code>t_rcv(3N)</code> or <code>t_rcvv(3N)</code>. Such an attempt will fail with <i>t_error</i> set to <code>TOUTSTATE</code>. However, the user may continue to send data over the connection if <code>t_sndrel(3N)</code> or <code>t_sndreldata(3N)</code> has not been called by the user.</p> <p>The field <i>reason</i> specifies the reason for the disconnection through a protocol-dependent <i>reason code</i>, and <i>udata</i> identifies any user data that was sent with the disconnection; the field <i>sequence</i> is not used.</p> <p>If a user does not care if there is incoming data and does not need to know the value of <i>reason</i>, <i>discon</i> may be a null pointer, and any user data associated with the disconnection will be discarded.</p> <p>If <code>discon->udata.maxlen</code> is greater than zero and less than the length of the value, <code>t_rcvreldata()</code> fails with <code>t_errno</code> set to <code>TBUFOVFLW</code>.</p> <p>This function is an optional service of the transport provider, only supported by providers of service type <code>T_COTS_ORD</code>. The flag <code>T_ORDRELDATA</code> in the <i>info->flag</i> field returned by <code>t_open(3N)</code> or <code>t_getinfo(3N)</code> indicates that the provider supports orderly release user data; when the flag is not set, this function behaves like <code>t_rcvrel(3N)</code> and no user data is returned.</p> <p>This function may not be available on all systems.</p> |
| RETURN VALUES | Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and <code>t_errno</code> is set to indicate an error. |
| VALID STATES | <code>T_DATAXFER</code> , <code>T_OUTREL</code> . |

ERRORS

On failure, `t_errno` is set to one of the following:

| | |
|-------------|--|
| TBADF | The specified file descriptor does not refer to a transport endpoint. |
| TBUFOVFLW | The number of bytes allocated for incoming data (<code>maxlen</code>) is greater than 0 but not sufficient to store the data, and the disconnection information to be returned in <i>discon</i> will be discarded. The provider state, as seen by the user, will be changed as if the data was successfully retrieved. |
| TLOOK | An asynchronous event has occurred on this transport endpoint and requires immediate attention. |
| TNOREL | No orderly release indication currently exists on the specified transport endpoint. |
| TNOTSUPPORT | Orderly release is not supported by the underlying transport provider. |
| TOUTSTATE | The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. |
| TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). |
| TSYSERR | A system error has occurred during execution of this function. |

**TLI
COMPATIBILITY
ATTRIBUTES**

In the TLI interface definition, no counterpart of this routine was defined.

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`t_getinfo(3N)`, `t_open(3N)`, `t_sndreldata(3N)`, `t_rcvrel(3N)`, `t_sndrel(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

NOTES

The interfaces `t_sndreldata(3N)` and `t_rcvreldata()` are only for use with a specific transport called "minimal OSI," which is not available on the Solaris

platform. These interfaces are not available for use in conjunction with Internet Transports (TCP or UDP).

| | |
|--------------------|---|
| NAME | t_rcvudata – receive a data unit |
| SYNOPSIS | <pre>#include <xti.h> int t_rcvudata(int fd, struct t_unitdata *unitdata, int *flags);</pre> |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>This function is used in connectionless-mode to receive a data unit from another transport user. The argument <code>fd</code> identifies the local transport endpoint through which data will be received, <code>unitdata</code> holds information associated with the received data unit, and <code>flags</code> is set on return to indicate that the complete data unit was not received. The argument <code>unitdata</code> points to a <code>t_unitdata</code> structure containing the following members:</p> <pre>struct netbuf addr; struct netbuf opt; struct netbuf udata;</pre> <p>The <code>maxlen</code> field of <code>addr</code>, <code>opt</code> and <code>udata</code> must be set before calling this function to indicate the maximum size of the buffer for each. If the <code>maxlen</code> field of <code>addr</code> or <code>opt</code> is set to zero, no information is returned in the <code>buf</code> field of this parameter.</p> <p>On return from this call, <code>addr</code> specifies the protocol address of the sending user, <code>opt</code> identifies options that were associated with this data unit, and <code>udata</code> specifies the user data that was received.</p> <p>By default, <code>t_rcvudata()</code> operates in synchronous mode and will wait for a data unit to arrive if none is currently available. However, if <code>O_NONBLOCK</code> is set by means of <code>t_open(3N)</code> or <code>fcntl(2)</code>, <code>t_rcvudata()</code> will execute in asynchronous mode and will fail if no data units are available.</p> <p>If the buffer defined in the <code>udata</code> field of <code>unitdata</code> is not large enough to hold the current data unit, the buffer will be filled and <code>T_MORE</code> will be set in <code>flags</code> on return to indicate that another <code>t_rcvudata()</code> should be called to retrieve the rest of the data unit. Subsequent calls to <code>t_rcvudata()</code> will return zero for the length of the address and options until the full data unit has been received.</p> |

| | | | | | | | | | | | | | | | | | |
|--------------------------|---|-------|---|-----------|--|-------|---|---------|--|-------------|--|-----------|--|--------|---|---------|--|
| | If the call is interrupted, t_rcvudata() will return EINTR and no datagrams will have been removed from the endpoint. | | | | | | | | | | | | | | | | |
| RETURN VALUES | Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and <code>t_errno</code> is set to indicate an error. | | | | | | | | | | | | | | | | |
| VALID STATES | T_IDLE. | | | | | | | | | | | | | | | | |
| ERRORS | On failure, <code>t_errno</code> is set to one of the following: | | | | | | | | | | | | | | | | |
| | <table border="0"> <tr> <td>TBADF</td> <td>The specified file descriptor does not refer to a transport endpoint.</td> </tr> <tr> <td>TBUFOVFLW</td> <td>The number of bytes allocated for the incoming protocol address or options (<i>maxlen</i>) is greater than 0 but not sufficient to store the information. The unit data information to be returned in <i>unitdata</i> will be discarded.</td> </tr> <tr> <td>TLOOK</td> <td>An asynchronous event has occurred on this transport endpoint and requires immediate attention.</td> </tr> <tr> <td>TNODATA</td> <td>O_NONBLOCK was set, but no data units are currently available from the transport provider.</td> </tr> <tr> <td>TNOTSUPPORT</td> <td>This function is not supported by the underlying transport provider.</td> </tr> <tr> <td>TOUTSTATE</td> <td>The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid.</td> </tr> <tr> <td>TPROTO</td> <td>This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>).</td> </tr> <tr> <td>TSYSERR</td> <td>A system error has occurred during execution of this function.</td> </tr> </table> | TBADF | The specified file descriptor does not refer to a transport endpoint. | TBUFOVFLW | The number of bytes allocated for the incoming protocol address or options (<i>maxlen</i>) is greater than 0 but not sufficient to store the information. The unit data information to be returned in <i>unitdata</i> will be discarded. | TLOOK | An asynchronous event has occurred on this transport endpoint and requires immediate attention. | TNODATA | O_NONBLOCK was set, but no data units are currently available from the transport provider. | TNOTSUPPORT | This function is not supported by the underlying transport provider. | TOUTSTATE | The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. | TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). | TSYSERR | A system error has occurred during execution of this function. |
| TBADF | The specified file descriptor does not refer to a transport endpoint. | | | | | | | | | | | | | | | | |
| TBUFOVFLW | The number of bytes allocated for the incoming protocol address or options (<i>maxlen</i>) is greater than 0 but not sufficient to store the information. The unit data information to be returned in <i>unitdata</i> will be discarded. | | | | | | | | | | | | | | | | |
| TLOOK | An asynchronous event has occurred on this transport endpoint and requires immediate attention. | | | | | | | | | | | | | | | | |
| TNODATA | O_NONBLOCK was set, but no data units are currently available from the transport provider. | | | | | | | | | | | | | | | | |
| TNOTSUPPORT | This function is not supported by the underlying transport provider. | | | | | | | | | | | | | | | | |
| TOUTSTATE | The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. | | | | | | | | | | | | | | | | |
| TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). | | | | | | | | | | | | | | | | |
| TSYSERR | A system error has occurred during execution of this function. | | | | | | | | | | | | | | | | |
| TLI COMPATIBILITY | The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below. | | | | | | | | | | | | | | | | |
| Interface Header | The XTI interfaces use the header file, <code>xti.h</code> . TLI interfaces should <i>not</i> use this header. They should use the header: <code>#include<tiuser.h></code> | | | | | | | | | | | | | | | | |

Error Description Values

The `t_errno` values that can be set by the XTI interface and cannot be set by the TLI interface are:

TPROTO

TOUTSTATE

A `t_errno` value that this routine can return under different circumstances than its XTI counterpart is `TBUFOVFLW`. It can be returned even when the `maxlen` field of the corresponding buffer has been set to zero.

Option Buffers

The format of the options in an `opt` buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not fix the buffer format.

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`fcntl(2)`, `t_alloc(3N)`, `t_open(3N)`, `t_rcvuderr(3N)`, `t_sndudata(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

| | |
|----------------------|---|
| NAME | t_rcvuderr – receive a unit data error indication |
| SYNOPSIS | <pre>#include <xti.h> int t_rcvuderr(int fd, struct t_uderr *uderr);</pre> |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>This function is used in connectionless-mode to receive information concerning an error on a previously sent data unit, and should only be issued following a unit data error indication. It informs the transport user that a data unit with a specific destination address and protocol options produced an error. The argument <code>fd</code> identifies the local transport endpoint through which the error report will be received, and <code>uderr</code> points to a <code>t_uderr</code> structure containing the following members:</p> <pre>struct netbuf addr; struct netbuf opt; t_scalar_t error;</pre> <p>The <code>maxlen</code> field of <code>addr</code> and <code>opt</code> must be set before calling this function to indicate the maximum size of the buffer for each. If this field is set to zero for <code>addr</code> or <code>opt</code>, no information is returned in the <code>buf</code> field of this parameter.</p> <p>On return from this call, the <code>addr</code> structure specifies the destination protocol address of the erroneous data unit, the <code>opt</code> structure identifies options that were associated with the data unit, and <code>error</code> specifies a protocol-dependent error code.</p> <p>If the user does not care to identify the data unit that produced an error, <code>uderr</code> may be set to a null pointer, and <code>t_rcvuderr()</code> will simply clear the error indication without reporting any information to the user.</p> |
| RETURN VALUES | Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and <code>t_errno</code> is set to indicate an error. |
| VALID STATES | T_IDLE. |
| ERRORS | On failure, <code>t_errno</code> is set to one of the following: |

| | | |
|---------------------------------|-------------|---|
| | TBADF | The specified file descriptor does not refer to a transport endpoint. |
| | TBUFOVFLW | The number of bytes allocated for the incoming protocol address or options (<i>maxlen</i>) is greater than 0 but not sufficient to store the information. The unit data error information to be returned in <i>uderr</i> will be discarded. |
| | TNOTSUPPORT | This function is not supported by the underlying transport provider. |
| | TNOUDERR | No unit data error indication currently exists on the specified transport endpoint. |
| | TOUTSTATE | The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. |
| | TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<i>t_errno</i>). |
| | TSYSERR | A system error has occurred during execution of this function. |
| TLI COMPATIBILITY | | The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below. |
| Interface Header | | The XTI interfaces use the header file, <i>xti.h</i> . TLI interfaces should <i>not</i> use this header. They should use the header: <pre>#include <tiuser.h></pre> |
| Error Description Values | | The <i>t_errno</i> values TPROTO and TOUTSTATE can be set by the XTI interface but not by the TLI interface. A <i>t_errno</i> value that this routine can return under different circumstances than its XTI counterpart is TBUFOVFLW. It can be returned even when the <i>maxlen</i> field of the corresponding buffer has been set to zero. |
| Option Buffers | | The format of the options in an <i>opt</i> buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not fix the buffer format. For more information refer to the <i>Transport Interfaces Programming Guide</i> |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

t_rcvudata(3N), **t_sndudata(3N)**, **attributes(5)**

Transport Interfaces Programming Guide

| | |
|--------------------|--|
| NAME | t_rcvv – receive data or expedited data sent over a connection and put the data into one or more non-contiguous buffers |
| SYNOPSIS | <pre>#include <xti.h> int t_rcvv(int fd, struct t_iovec *iov, unsigned int iovcount, int *flags);</pre> |
| DESCRIPTION | <p>This function receives either normal or expedited data. The argument <i>fd</i> identifies the local transport endpoint through which data will arrive, <i>iov</i> points to an array of buffer address/buffer size pairs (<i>iov_base</i>, <i>iov_len</i>). The t_rcvv() function receives data into the buffers specified by <i>iov0.iov_base</i>, <i>iov1.iov_base</i>, through <i>iov [iovcount-1].iov_base</i>, always filling one buffer before proceeding to the next.</p> <p>Note that the limit on the total number of bytes available in all buffers passed:</p> $iov(0).iov_len + \dots + iov(iovcount-1).iov_len$ <p>may be constrained by implementation limits. If no other constraint applies, it will be limited by <code>INT_MAX</code>. In practice, the availability of memory to an application is likely to impose a lower limit on the amount of data that can be sent or received using scatter/gather functions.</p> <p>The argument <i>iovcount</i> contains the number of buffers which is limited to <code>T_IOV_MAX</code>, which is an implementation-defined value of at least 16. If the limit is exceeded, the function will fail with <code>TBADDATA</code>.</p> <p>The argument <i>flags</i> may be set on return from t_rcvv() and specifies optional flags as described below.</p> <p>By default, t_rcvv() operates in synchronous mode and will wait for data to arrive if none is currently available. However, if <code>O_NONBLOCK</code> is set by means of t_open(3N) or fcntl(2), t_rcvv() will execute in asynchronous mode and will fail if no data is available. See <code>TNODATA</code> below.</p> <p>On return from the call, if <code>T_MORE</code> is set in <i>flags</i>, this indicates that there is more data, and the current transport service data unit (TSDU) or expedited transport service data unit (ETSDU) must be received in multiple t_rcvv() or t_rcv(3N) calls. In the asynchronous mode, or under unusual conditions (for example, the arrival of a signal or <code>T_EXDATA</code> event), the <code>T_MORE</code> flag may be set on return from the t_rcvv() call even when the number of bytes received is less than the total size of all the receive buffers. Each t_rcvv() with the <code>T_MORE</code> flag set indicates that another t_rcvv() must follow to get more data for the current TSDU. The end of the TSDU is identified by the return of a t_rcvv()</p> |

call with the `T_MORE` flag not set. If the transport provider does not support the concept of a TSDU as indicated in the *info* argument on return from `t_open(3N)` or `t_getinfo(3N)`, the `T_MORE` flag is not meaningful and should be ignored. If the amount of buffer space passed in *iov* is greater than zero on the call to `t_rcvv()`, then `t_rcvv()` will return 0 only if the end of a TSDU is being returned to the user.

On return, the data is expedited if `T_EXPEDITED` is set in flags. If `T_MORE` is also set, it indicates that the number of expedited bytes exceeded *nbytes*, a signal has interrupted the call, or that an entire ETSDU was not available (only for transport protocols that support fragmentation of ETSDUs). The rest of the ETSDU will be returned by subsequent calls to `t_rcvv()` which will return with `T_EXPEDITED` set in flags. The end of the ETSDU is identified by the return of a `t_rcvv()` call with `T_EXPEDITED` set and `T_MORE` cleared. If the entire ETSDU is not available it is possible for normal data fragments to be returned between the initial and final fragments of an ETSDU.

If a signal arrives, `t_rcvv()` returns, giving the user any data currently available. If no data is available, `t_rcvv()` returns -1, sets `t_errno` to `TSYSERR` and `errno` to `EINTR`. If some data is available, `t_rcvv()` returns the number of bytes received and `T_MORE` is set in flags.

In synchronous mode, the only way for the user to be notified of the arrival of normal or expedited data is to issue this function or check for the `T_DATA` or `T_EXDATA` events using the `t_lock(3N)` function. Additionally, the process can arrange to be notified via the EM interface.

RETURN VALUES

On successful completion, `t_rcvv()` returns the number of bytes received. Otherwise, it returns -1 on failure and `t_errno` is set to indicate the error.

VALID STATES

`T_DATAXFER`, `T_OUTREL`.

ERRORS

On failure, `t_errno` is set to one of the following:

| | |
|--------------------------|--|
| <code>TBADDATA</code> | <i>iovcount</i> is greater than <code>T_IOV_MAX</code> . |
| <code>TBADF</code> | The specified file descriptor does not refer to a transport endpoint. |
| <code>TLOOK</code> | An asynchronous event has occurred on this transport endpoint and requires immediate attention. |
| <code>TNODATA</code> | <code>O_NONBLOCK</code> was set, but no data is currently available from the transport provider. |
| <code>TNOTSUPPORT</code> | This function is not supported by the underlying transport provider. |

| | |
|-----------|---|
| TOUTSTATE | The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. |
| TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<i>t_errno</i>). |
| TSYSERR | A system error has occurred during execution of this function. |

**TLI
COMPATIBILITY
ATTRIBUTES**

In the TLI interface definition, no counterpart of this routine was defined.

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

fcntl(2), **t_getinfo(3N)**, **t_lock(3N)**, **t_open(3N)**, **t_rcv(3N)**, **t_snd(3N)**, **t_sndv(3N)**, **attributes(5)**

Transport Interfaces Programming Guide

| | |
|--------------------|--|
| NAME | t_rcvvudata – receive a data unit into one or more noncontiguous buffers |
| SYNOPSIS | <pre>#include <xti.h></pre> <pre>int t_rcvvudata(int fd, struct t_unitdata *unitdata, struct t_iovec *iov, unsigned int iovcount, int *flags);</pre> |
| DESCRIPTION | <p>This function is used in connectionless mode to receive a data unit from another transport user. The argument <i>fd</i> identifies the local transport endpoint through which data will be received, <i>unitdata</i> holds information associated with the received data unit, <i>iovcount</i> contains the number of non-contiguous udata buffers which is limited to T_IOV_MAX, which is an implementation-defined value of at least 16, and <i>flags</i> is set on return to indicate that the complete data unit was not received. If the limit on <i>iovcount</i> is exceeded, the function fails with TBADDDATA. The argument <i>unitdata</i> points to a t_unitdata structure containing the following members:</p> <pre>struct netbuf addr; struct netbuf opt; struct netbuf udata;</pre> <p>The <i>maxlen</i> field of <i>addr</i> and <i>opt</i> must be set before calling this function to indicate the maximum size of the buffer for each. The <i>udata</i> field of t_unitdata is not used. The <i>iov_len</i> and <i>iov_base</i> fields of "<i>iov</i>" through <i>iov</i> [<i>iovcount</i>-1] must be set before calling t_rcvvudata() to define the buffer where the userdata will be placed. If the maxlen field of <i>addr</i> or <i>opt</i> is set to zero then no information is returned in the <i>buf</i> field for this parameter.</p> <p>On return from this call, <i>addr</i> specifies the protocol address of the sending user, <i>opt</i> identifies options that were associated with this data unit, and <i>iov</i>[0].<i>iov_base</i> through <i>iov</i> [<i>iovcount</i>-1].<i>iov_base</i> contains the user data that was received. The return value of t_rcvvudata() is the number of bytes of user data given to the user.</p> <p>Note that the limit on the total number of bytes available in all buffers passed:</p> <pre>iov(0).iov_len + . . . + iov(iovcount-1).iov_len)</pre> <p>may be constrained by implementation limits. If no other constraint applies, it will be limited by INT_MAX. In practice, the availability of memory to an application is likely to impose a lower limit on the amount of data that can be sent or received using scatter/gather functions.</p> |

By default, **t_rcvvudata()** operates in synchronous mode and waits for a data unit to arrive if none is currently available. However, if `O_NONBLOCK` is set by means of **t_open(3N)** or **fcntl(2)**, **t_rcvvudata()** executes in asynchronous mode and fails if no data units are available.

If the buffers defined in the *iov[]* array are not large enough to hold the current data unit, the buffers will be filled and `T_MORE` will be set in flags on return to indicate that another **t_rcvvudata()** should be called to retrieve the rest of the data unit. Subsequent calls to **t_rcvvudata()** will return zero for the length of the address and options, until the full data unit has been received.

RETURN VALUES

On successful completion, **t_rcvvudata()** returns the number of bytes received. Otherwise, it returns `-1` on failure and `t_errno` is set to indicate the error.

VALID STATES

`T_IDLE`.

ERRORS

On failure, `t_errno` is set to one of the following:

| | |
|--------------------------|--|
| <code>TBADDATA</code> | <i>iovcount</i> is greater than <code>T_IOV_MAX</code> . |
| <code>TBADF</code> | The specified file descriptor does not refer to a transport endpoint. |
| <code>TBUFOVFLW</code> | The number of bytes allocated for the incoming protocol address or options (<i>maxlen</i>) is greater than 0 but not sufficient to store the information. The unit data information to be returned in <i>unitdata</i> will be discarded. |
| <code>TLOOK</code> | An asynchronous event has occurred on this transport endpoint and requires immediate attention. |
| <code>TNODATA</code> | <code>O_NONBLOCK</code> was set, but no data units are currently available from the transport provider. |
| <code>TNOTSUPPORT</code> | This function is not supported by the underlying transport provider. |
| <code>TOUTSTATE</code> | The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. |
| <code>TPROTO</code> | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). |
| <code>TSYSERR</code> | A system error has occurred during execution of this function. |

**TLI
COMPATIBILITY****ATTRIBUTES**

In the TLI interface definition, no counterpart of this routine was defined.

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

fcntl(2), **t_alloc(3N)**, **t_open(3N)**, **t_rcvvudata(3N)**,
t_rcvuderr(3N), **t_sndudata(3N)**, **t_sndvudata(3N)**, **attributes(5)**

Transport Interfaces Programming Guide

| | |
|----------------------|--|
| NAME | truncate, ftruncate – set a file to a specified length |
| SYNOPSIS | <pre>#include <unistd.h> int truncate(const char * path, off_t length); int ftruncate(int fildes, off_t length);</pre> |
| DESCRIPTION | <p>The truncate() function causes the regular file named by <i>path</i> to have a size of <i>length</i> bytes.</p> <p>The ftruncate() function causes the regular file referenced by <i>fildes</i> to have a size of <i>length</i> bytes.</p> <p>The effect of ftruncate() and truncate() on other types of files is unspecified. If the file previously was larger than <i>length</i>, the extra data is lost. If it was previously shorter than <i>length</i>, bytes between the old and new lengths are read as zeroes. With ftruncate(), the file must be open for writing; for truncate(), the process must have write permission for the file.</p> <p>If the request would cause the file size to exceed the soft file size limit for the process, the request will fail and the implementation will generate the SIGXFSZ signal for the process.</p> <p>These functions do not modify the file offset for any open file descriptions associated with the file. On successful completion, if the file size is changed, these functions will mark for update the <i>st_ctime</i> and <i>st_mtime</i> fields of the file, and if the file is a regular file, the <i>S_ISUID</i> and <i>S_ISGID</i> bits of the file mode may be cleared.</p> |
| RETURN VALUES | Upon successful completion, ftruncate() and truncate() return 0. Otherwise, -1 is returned and <i>errno</i> is set to indicate the error. |
| ERRORS | <p>The ftruncate() and truncate() functions will fail if:</p> <p>EINTR A signal was caught during execution.</p> <p>EINVAL The <i>length</i> argument was less than 0.</p> <p>EFBIG or EINVAL The <i>length</i> argument was greater than the maximum file size.</p> <p>EIO An I/O error occurred while reading from or writing to a file system.</p> <p>The truncate() function will fail if:</p> <p>EACCES A component of the path prefix denies search permission, or write permission is denied on the file.</p> |

| | |
|---|---|
| EFAULT | The <i>path</i> argument points outside the process' allocated address space. |
| EINVAL | The <i>path</i> argument is not an ordinary file. |
| EISDIR | The named file is a directory. |
| ELOOP | Too many symbolic links were encountered in resolving <i>path</i> . |
| EMFILE | The maximum number of file descriptors available to the process has been reached. |
| ENAMETOOLONG | The length of the specified pathname exceeds <code>PATH_MAX</code> bytes, or the length of a component of the pathname exceeds <code>NAME_MAX</code> bytes. |
| ENOENT | A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string. |
| ENFILE | Additional space could not be allocated for the system file table. |
| ENOTDIR | A component of the path prefix of <i>path</i> is not a directory. |
| ENOLINK | The <i>path</i> argument points to a remote machine and the link to that machine is no longer active. |
| EROFS The <code>ftruncate()</code> function will fail if: | The named file resides on a read-only file system. |
| EAGAIN | The file exists, mandatory file/record locking is set, and there are outstanding record locks on the file (see <code>chmod(2)</code>). |
| EBADF or EINVAL | The <i>fildev</i> argument is not a file descriptor open for writing. |
| EFBIG | The file is a regular file and <i>length</i> is greater than the offset maximum established in the open file description associated with <i>fildev</i> . |
| EINVAL | The <i>fildev</i> argument references a file that was opened without write permission. |

EINVAL The *fildev* argument does not correspond to an ordinary file.

ENOLINK The *fildev* argument points to a remote machine and the link to that machine is no longer active.

The **truncate()** function may fail if:

ENAMETOOLONG Pathname resolution of a symbolic link produced an intermediate result whose

USAGE The **truncate()** and **ftruncate()** functions have transitional interfaces for 64-bit file offsets. See **lf64(5)**.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **chmod(2)**, **fcntl(2)**, **open(2)**, **attributes(5)**, **lf64(5)**

| | |
|--------------------|---|
| NAME | tsearch, tfind, tdelete, twalk – manage binary search trees |
| SYNOPSIS | <pre>#include <search.h> void * tsearch(const void * <i>key</i>, void ** <i>rootp</i>, int (* <i>compar</i>)(const void *, const void *)); void * tfind(const void * <i>key</i>, void * const * <i>rootp</i>, int (* <i>compar</i>)(const void *, const void *)); void * tdelete(const void * <i>key</i>, void ** <i>rootp</i>, int (* <i>compar</i>)(const void *, const void *)); void twalk(const void * <i>root</i>, void(* <i>action</i>)(void *, VISIT, int));</pre> |
| DESCRIPTION | <p>The tsearch() , tfind() , tdelete() , and twalk() functions are routines for manipulating binary search trees. They are generalized from <i>Knuth (6.2.2) Algorithms T and D</i>. All comparisons are done with a user-supplied routine. This routine is called with two arguments, the pointers to the elements being compared. It returns an integer less than, equal to, or greater than 0, according to whether the first argument is to be considered less than, equal to or greater than the second argument. The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.</p> <p>The tsearch() function is used to build and access the tree. The <i>key</i> argument is a pointer to a datum to be accessed or stored. If there is a datum in the tree equal to <i>*key</i> (the value pointed to by <i>key</i>), a pointer to this found datum is returned. Otherwise, <i>*key</i> is inserted, and a pointer to it returned. Only pointers are copied, so the calling routine must store the data. The <i>rootp</i> argument points to a variable that points to the root of the tree. A null value for the variable pointed to by <i>rootp</i> denotes an empty tree; in this case, the variable will be set to point to the datum which will be at the root of the new tree.</p> <p>Like tsearch() , tfind() will search for a datum in the tree, returning a pointer to it if found. However, if it is not found, tfind() will return a null pointer. The arguments for tfind() are the same as for tsearch() .</p> <p>The tdelete() function deletes a node from a binary search tree. The arguments are the same as for tsearch() . The variable pointed to by <i>rootp</i> will be changed if the deleted node was the root of the tree. tdelete() returns a pointer to the parent of the deleted node, or a null pointer if the node is not found.</p> <p>The twalk() function traverses a binary search tree. The <i>root</i> argument is the root of the tree to be traversed. (Any node in a tree may be used as the root for a walk below that node.) <i>action</i> is the name of a routine to be invoked at each node. This routine is, in turn, called with three arguments. The first argument is the address of the node being visited. The second argument is a value from an enumeration data type</p> |

```
typedef enum { preorder, postorder, endorder, leaf } VISIT;
```

(defined in `<search.h>`), depending on whether this is the first, second or third time that the node has been visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a leaf. The third argument is the level of the node in the tree, with the root being level zero.

The pointers to the key and the root of the tree should be of type pointer-to-element, and cast to type pointer-to-character. Similarly, although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

RETURN VALUES

If the node is found, both `tsearch()` and `tfind()` return a pointer to it. If not, `tfind()` returns a null pointer, and `tsearch()` returns a pointer to the inserted item.

A null pointer is returned by `tsearch()` if there is not enough space available to create a new node.

A null pointer is returned by `tsearch()` , `tfind()` and `tdelete()` if `rootp` is a null pointer on entry.

The `tdelete()` function returns a pointer to the parent of the deleted node, or a null pointer if the node is not found.

The `twalk()` function returns no value.

ERRORS

No errors are defined.

USAGE

The `root` argument to `twalk()` is one level of indirection less than the `rootp` arguments to `tsearch()` and `tdelete()` .

There are two nomenclatures used to refer to the order in which tree nodes are visited. `tsearch()` uses preorder, postorder and endorder to refer respectively to visiting a node before any of its children, after its left child and before its right, and after both its children. The alternate nomenclature uses preorder, inorder and postorder to refer to the same visits, which could result in some confusion over the meaning of postorder.

If the calling function alters the pointer to the root, results are unpredictable.

EXAMPLES

EXAMPLE 1 A sample program of using `tsearch` function.

The following code reads in strings and stores structures containing a pointer to each string and a count of its length. It then walks the tree, printing out the stored strings and their lengths in alphabetical order.

```
#include <string.h>
#include <stdio.h>
```

```

#include <search.h>
struct node {
    char *string;
    int length;
};
char string_space[10000];
struct node nodes[500];
void *root = NULL;

int node_compare(const void *node1, const void *node2) {
    return strcmp(((const struct node *) node1)⇒string,
                 ((const struct node *) node2)⇒string);
}

void print_node(void *node, VISIT order, int level) {
    if (order == preorder || order == leaf) {
        printf("length=%d, string=%20s\\",
              (*(struct node **)node)⇒length,
              (*(struct node **)node)⇒string);
    }
}

main()
{
    char *strptr = string_space;
    struct node *nodeptr = nodes;
    int i = 0;

    while (gets(strptr) != NULL && i++ < 500) {
        nodeptr⇒string = strptr;
        nodeptr⇒length = strlen(strptr);
        (void) tsearch((void *)nodeptr,
                     &root, node_compare);
        strptr += nodeptr⇒length + 1;
        nodeptr++;
    }
    twalk(root, print_node);
}

```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

bsearch(3C), **hsearch(3C)**, **lsearch(3C)**, **attributes(5)**

| | |
|--------------------|---|
| NAME | t_snd – send data or expedited data over a connection |
| SYNOPSIS | <pre>#include <xti.h> int t_snd(int fd, void *buf, unsigned int nbytes, int flags);</pre> |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>This function is used to send either normal or expedited data. The argument <i>fd</i> identifies the local transport endpoint over which data should be sent, <i>buf</i> points to the user data, <i>nbytes</i> specifies the number of bytes of user data to be sent, and <i>flags</i> specifies any optional flags described below:</p> <p>T_EXPEDITED If set in <i>flags</i>, the data will be sent as expedited data and will be subject to the interpretations of the transport provider.</p> <p>T_MORE If set in <i>flags</i>, this indicates to the transport provider that the transport service data unit (TSDU) (or expedited transport service data unit - ETSDU) is being sent through multiple <code>t_snd()</code> calls. Each <code>t_snd()</code> with the <code>T_MORE</code> flag set indicates that another <code>t_snd()</code> will follow with more data for the current TSDU (or ETSDU).</p> <p>The end of the TSDU (or ETSDU) is identified by a <code>t_snd()</code> call with the <code>T_MORE</code> flag not set. Use of <code>T_MORE</code> enables a user to break up large logical data units without losing the boundaries of those units at the other end of the connection. The flag implies nothing about how the data is packaged for transfer below the transport interface. If the transport provider does not support the concept of a TSDU as indicated in the <i>info</i> argument on return from <code>t_open(3N)</code> or <code>t_getinfo(3N)</code>, the <code>T_MORE</code> flag is not meaningful and will be ignored if set.</p> <p>The sending of a zero-length fragment of a TSDU or ETSDU is only permitted where this is used to indicate the end of a TSDU or ETSDU; that is, when the <code>T_MORE</code> flag is not set. Some transport providers also forbid zero-length TSDUs and ETSDUs.</p> <p>T_PUSH If set in <i>flags</i>, requests that the provider transmit all data that it has accumulated but not sent. The request is a local action</p> |

on the provider and does not affect any similarly named protocol flag (for example, the TCP PUSH flag). This effect of setting this flag is protocol-dependent, and it may be ignored entirely by transport providers which do not support the use of this feature.

Note that the communications provider is free to collect data in a send buffer until it accumulates a sufficient amount for transmission.

By default, **t_snd()** operates in synchronous mode and may wait if flow control restrictions prevent the data from being accepted by the local transport provider at the time the call is made. However, if `O_NONBLOCK` is set by means of **t_open(3N)** or **fcntl(2)**, **t_snd()** will execute in asynchronous mode, and will fail immediately if there are flow control restrictions. The process can arrange to be informed when the flow control restrictions are cleared by means of either **t_look(3N)** or the EM interface.

On successful completion, **t_snd()** returns the number of bytes (octets) accepted by the communications provider. Normally this will equal the number of octets specified in `nbytes`. However, if `O_NONBLOCK` is set or the function is interrupted by a signal, it is possible that only part of the data has actually been accepted by the communications provider. In this case, **t_snd()** returns a value that is less than the value of `nbytes`. If **t_snd()** is interrupted by a signal before it could transfer data to the communications provider, it returns `-1` with `t_errno` set to `TSYSERR` and `errno` set to `EINTR`.

If `nbytes` is zero and sending of zero bytes is not supported by the underlying communications service, **t_snd()** returns `-1` with `t_errno` set to `TBADDATA`.

The size of each TSDU or ETSDU must not exceed the limits of the transport provider as specified by the current values in the TSDU or ETSDU fields in the *info* argument returned by **t_getinfo(3N)**.

The error `TLOOK` is returned for asynchronous events. It is required only for an incoming disconnect event but may be returned for other events.

RETURN VALUES

On successful completion, **t_snd()** returns the number of bytes accepted by the transport provider. Otherwise, `-1` is returned on failure and `t_errno` is set to indicate the error.

Note that if the number of bytes accepted by the communications provider is less than the number of bytes requested, this may either indicate that `O_NONBLOCK` is set and the communications provider is blocked due to flow control, or that `O_NONBLOCK` is clear and the function was interrupted by a signal.

ERRORS

On failure, `t_errno` is set to one of the following:

`TBADDATA` Illegal amount of data:

- A single send was attempted specifying a TSDU (ETSDU) or fragment TSDU (ETSDU) greater than that specified by the current values of the TSDU or ETSDU fields in the *info* argument.
- A send of a zero byte TSDU (ETSDU) or zero byte fragment of a TSDU (ETSDU) is not supported by the provider.
- Multiple sends were attempted resulting in a TSDU (ETSDU) larger than that specified by the current value of the TSDU or ETSDU fields in the *info* argument – the ability of an XTI implementation to detect such an error case is implementation-dependent. See WARNINGS, below.

| | |
|-------------|---|
| TBADF | The specified file descriptor does not refer to a transport endpoint. |
| TBADFLAG | An invalid flag was specified. |
| TFLOW | O_NONBLOCK was set, but the flow control mechanism prevented the transport provider from accepting any data at this time. |
| TLOOK | An asynchronous event has occurred on this transport endpoint. |
| TNOTSUPPORT | This function is not supported by the underlying transport provider. |
| TOUTSTATE | The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. |
| TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<i>t_errno</i>). |
| TSYSERR | A system error has occurred during execution of this function. |

TLI COMPATIBILITY

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, *xti.h*. TLI interfaces should *not* use this header. They should use the header:

```
#include <tiuser.h>
```

**Error Description
Values**

The `t_errno` values that can be set by the XTI interface and cannot be set by the TLI interface are:

```
TPROTO
TLOOK
TBADFLAG
TOUTSTATE
```

The `t_errno` values that this routine can return under different circumstances than its XTI counterpart are:

```
TBADDATA
```

In the `TBADDATA` error cases described above, `TBADDATA` is returned, only for illegal zero byte `TSDU` (`ETSDU`) send attempts.

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`fcntl(2)`, `t_getinfo(3N)`, `t_look(3N)`, `t_open(3N)`, `t_rcv(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

WARNINGS

It is important to remember that the transport provider treats all users of a transport endpoint as a single user. Therefore if several processes issue concurrent `t_snd()` calls then the different data may be intermixed.

Multiple sends which exceed the maximum `TSDU` or `ETSDU` size may not be discovered by XTI. In this case an implementation-dependent error will result, generated by the transport provider, perhaps on a subsequent XTI call. This error may take the form of a connection abort, a `TSYSERR`, a `TBADDATA` or a `TPROTO` error.

If multiple sends which exceed the maximum TSDU or ETSDU size are detected by XTI, **t_snd()** fails with `TBADDATA`.

| | |
|----------------------|---|
| NAME | t_snddis – send user-initiated disconnection request |
| SYNOPSIS | <pre>#include <xti.h></pre> <pre>int t_snddis(int fd, const struct t_call *call);</pre> |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>This function is used to initiate an abortive release on an already established connection, or to reject a connection request. The argument <code>fd</code> identifies the local transport endpoint of the connection, and <code>call</code> specifies information associated with the abortive release. The argument <code>call</code> points to a <code>t_call</code> structure which contains the following members:</p> <pre> struct netbuf addr; struct netbuf opt; struct netbuf udata; int sequence; </pre> <p>The values in <code>call</code> have different semantics, depending on the context of the call to <code>t_snddis()</code>. When rejecting a connection request, <code>call</code> must be non-null and contain a valid value of <code>sequence</code> to uniquely identify the rejected connection indication to the transport provider. The <code>sequence</code> field is only meaningful if the transport connection is in the <code>T_INCON</code> state. The <code>addr</code> and <code>opt</code> fields of <code>call</code> are ignored. In all other cases, <code>call</code> need only be used when data is being sent with the disconnection request. The <code>addr</code>, <code>opt</code> and <code>sequence</code> fields of the <code>t_call</code> structure are ignored. If the user does not wish to send data to the remote user, the value of <code>call</code> may be a null pointer.</p> <p>The <code>udata</code> structure specifies the user data to be sent to the remote user. The amount of user data must not exceed the limits supported by the transport provider, as returned in the <code>discon</code> field, of the <code>info</code> argument of <code>t_open(3N)</code> or <code>t_getinfo(3N)</code>. If the <code>len</code> field of <code>udata</code> is zero, no data will be sent to the remote user.</p> |
| RETURN VALUES | Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and <code>t_errno</code> is set to indicate an error. |
| VALID STATES | <code>T_DATAXFER</code> , <code>T_OUTCON</code> , <code>T_OUTREL</code> , <code>T_INREL</code> , <code>T_INCON</code> (<code>ocnt > 0</code>). |

| | |
|---------------------------------|---|
| ERRORS | <p>On failure, <code>t_errno</code> is set to one of the following:</p> <p>TBADF The specified file descriptor does not refer to a transport endpoint.</p> <p>TBADDATA The amount of user data specified was not within the bounds allowed by the transport provider.</p> <p>TBADSEQ An invalid sequence number was specified, or a null <i>call</i> pointer was specified, when rejecting a connection request.</p> <p>TLOOK An asynchronous event, which requires attention, has occurred.</p> <p>TNOTSUPPORT This function is not supported by the underlying transport provider.</p> <p>TOUTSTATE The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid.</p> <p>TPROTO This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>).</p> <p>TSYSERR A system error has occurred during execution of this function.</p> |
| TLI COMPATIBILITY | <p>The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.</p> |
| Interface Header | <p>The XTI interfaces use the header file, <code>xti.h</code>. TLI interfaces should <i>not</i> use this header. They should use the header:</p> <pre>#include <tiuser.h></pre> |
| Error Description Values | <p>The <code>t_errno</code> value <code>TPROTO</code> can be set by the XTI interface but not by the TLI interface.</p> |
| Option Buffers | <p>The format of the options in an <code>opt</code> buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not fix the buffer format.</p> <p>For more information refer to the <i>Transport Interfaces Programming Guide</i></p> |
| ATTRIBUTES | <p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> |

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`t_connect(3N)`, `t_getinfo(3N)`, `t_listen(3N)`, `t_open(3N)`,
`t_send(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

WARNINGS

`t_snddis()` is an abortive disconnection. Therefore a `t_snddis()` issued on a connection endpoint may cause data previously sent by means of `t_snd(3N)`, or data not yet received, to be lost, even if an error is returned.

| | |
|----------------------|--|
| NAME | t_sndrel – initiate an orderly release |
| SYNOPSIS | #include <xti.h> int t_sndrel(int <i>fd</i>); |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>For transport providers of type <code>T_COTS_ORD</code>, this function is used to initiate an orderly release of the outgoing direction of data transfer and indicates to the transport provider that the transport user has no more data to send. The argument <i>fd</i> identifies the local transport endpoint where the connection exists. After calling <code>t_sndrel()</code>, the user may not send any more data over the connection. However, a user may continue to receive data if an orderly release indication has not been received. For transport providers of types other than <code>T_COTS_ORD</code>, this function fails with error <code>TNOTSUPPORT</code>.</p> |
| RETURN VALUES | Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and <code>t_errno</code> is set to indicate an error. |
| VALID STATES | <code>T_DATAXFER</code> , <code>T_INREL</code> . |
| ERRORS | On failure, <code>t_errno</code> is set to one of the following: |
| | <code>TBADF</code> The specified file descriptor does not refer to a transport endpoint. |
| | <code>TFLOW</code> <code>O_NONBLOCK</code> was set, but the flow control mechanism prevented the transport provider from accepting the function at this time. |
| | <code>TLOOK</code> An asynchronous event has occurred on this transport endpoint and requires immediate attention. |
| | <code>TNOTSUPPORT</code> This function is not supported by the underlying transport provider. |
| | <code>TOUTSTATE</code> The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. |

TPROTO This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (`t_errno`).

TSYSERR A system error has occurred during execution of this function.

TLI COMPATIBILITY

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```
#include <tiuser.h>
```

Error Description Values

The `t_errno` values that can be set by the XTI interface and cannot be set by the TLI interface are:

TPROTO

TLOOK

TOUTSTATE

Notes

Whenever this function fails with `t_error` set to `TFLOW`, `O_NONBLOCK` must have been set.

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`t_error(3N)`, `t_getinfo(3N)`, `t_open(3N)`, `t_rcvrel(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

| | |
|----------------------|---|
| NAME | t_sndreldata – initiate or respond to an orderly release with user data |
| SYNOPSIS | <pre>#include <xti.h> int t_sndreldata(int fd, struct t_discon *discon);</pre> |
| DESCRIPTION | <p>This function is used to initiate an orderly release of the outgoing direction of data transfer and to send user data with the release. The argument <i>fd</i> identifies the local transport endpoint where the connection exists, and <i>discon</i> points to a <i>t_discon</i> structure containing the following members:</p> <pre>struct netbuf udata; int reason; int sequence;</pre> <p>After calling t_sndreldata(), the user may not send any more data over the connection. However, a user may continue to receive data if an orderly release indication has not been received.</p> <p>The field <i>reason</i> specifies the reason for the disconnection through a protocol-dependent <i>reason code</i>, and <i>udata</i> identifies any user data that is sent with the disconnection; the field <i>sequence</i> is not used.</p> <p>The <i>udata</i> structure specifies the user data to be sent to the remote user. The amount of user data must not exceed the limits supported by the transport provider, as returned in the <i>discon</i> field of the <i>info</i> argument of t_open(3N) or t_getinfo(3N). If the <i>len</i> field of <i>udata</i> is zero or if the provider did not return T_ORDRELDATA in the t_open(3N) flags, no data will be sent to the remote user.</p> <p>If a user does not wish to send data and reason code to the remote user, the value of <i>discon</i> may be a null pointer.</p> <p>This function is an optional service of the transport provider, only supported by providers of service type T_COTS_ORD. The flag T_ORDRELDATA in the <i>info→flag</i> field returned by t_open(3N) or t_getinfo(3N) indicates that the provider supports orderly release user data.</p> <p>This function may not be available on all systems.</p> |
| RETURN VALUES | Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and <i>t_errno</i> is set to indicate an error. |
| VALID STATES | T_DATAXFER , T_INREL . |

ERRORS

On failure, `t_errno` is set to one of the following:

| | |
|-------------|--|
| TBADDATA | The amount of user data specified was not within the bounds allowed by the transport provider, or user data was supplied and the provider did not return <code>T_ORDRELDATA</code> in the <code>t_open(3N)</code> flags. |
| TBADF | The specified file descriptor does not refer to a transport endpoint. |
| TFLOW | <code>O_NONBLOCK</code> was set, but the flow control mechanism prevented the transport provider from accepting the function at this time. |
| TLOOK | An asynchronous event has occurred on this transport endpoint and requires immediate attention. |
| TNOTSUPPORT | Orderly release is not supported by the underlying transport provider. |
| TOUTSTATE | The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. |
| TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). |
| TSYSERR | A system error has occurred during execution of this function. |

**TLI
COMPATIBILITY
ATTRIBUTES**

In the TLI interface definition, no counterpart of this routine was defined.

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`t_getinfo(3N)`, `t_open(3N)`, `t_rcvrel(3N)`, `t_rcvreldata(3N)`, `t_sndrel(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

NOTES

The interfaces `t_sndreldata()` and `t_rcvreldata(3N)` are only for use with a specific transport called "minimal OSI," which is not available on the Solaris

platform. These interfaces are not available for use in conjunction with Internet Transports (TCP or UDP).

| | |
|--------------------|--|
| NAME | t_sndudata – send a data unit |
| SYNOPSIS | <pre>#include <xti.h></pre> <pre>int t_sndudata(int fd, const struct t_unitdata *unitdata);</pre> |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>This function is used in connectionless-mode to send a data unit to another transport user. The argument <code>fd</code> identifies the local transport endpoint through which data will be sent, and <code>unitdata</code> points to a <code>t_unitdata</code> structure containing the following members:</p> <pre>struct netbuf addr; struct netbuf opt; struct netbuf udata;</pre> <p>In <code>unitdata</code>, <code>addr</code> specifies the protocol address of the destination user, <code>opt</code> identifies options that the user wants associated with this request, and <code>udata</code> specifies the user data to be sent. The user may choose not to specify what protocol options are associated with the transfer by setting the <code>len</code> field of <code>opt</code> to zero. In this case, the provider uses the option values currently set for the communications endpoint.</p> <p>If the <code>len</code> field of <code>udata</code> is zero, and sending of zero octets is not supported by the underlying transport service, the <code>t_sndudata()</code> will return <code>-1</code> with <code>t_errno</code> set to <code>TBADDATA</code>.</p> <p>By default, <code>t_sndudata()</code> operates in synchronous mode and may wait if flow control restrictions prevent the data from being accepted by the local transport provider at the time the call is made. However, if <code>O_NONBLOCK</code> is set by means of <code>t_open(3N)</code> or <code>fcntl(2)</code>, <code>t_sndudata()</code> will execute in asynchronous mode and will fail under such conditions. The process can arrange to be notified of the clearance of a flow control restriction by means of either <code>t_look(3N)</code> or the EM interface.</p> <p>If the amount of data specified in <code>udata</code> exceeds the TSDU size as returned in the <code>tsdu</code> field of the <code>info</code> argument of <code>t_open(3N)</code> or <code>t_getinfo(3N)</code>, a <code>TBADDATA</code> error will be generated. If <code>t_sndudata()</code> is called before the</p> |

destination user has activated its transport endpoint (see `t_bind(3N)`), the data unit may be discarded.

If it is not possible for the transport provider to immediately detect the conditions that cause the errors `TBADADDR` and `TBADOPT`, these errors will alternatively be returned by `t_rcvuderr`. Therefore, an application must be prepared to receive these errors in both of these ways.

If the call is interrupted, `t_sndudata()` will return `EINTR` and the datagram will not be sent.

RETURN VALUES

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and `t_errno` is set to indicate an error.

VALID STATES

`T_IDLE`.

ERRORS

On failure, `t_errno` is set to one of the following:

| | |
|--------------------------|--|
| <code>TBADADDR</code> | The specified protocol address was in an incorrect format or contained illegal information. |
| <code>TBADDATA</code> | Illegal amount of data. A single send was attempted specifying a TSDU greater than that specified in the <i>info</i> argument, or a send of a zero byte TSDU is not supported by the provider. |
| <code>TBADF</code> | The specified file descriptor does not refer to a transport endpoint. |
| <code>TBADOPT</code> | The specified options were in an incorrect format or contained illegal information. |
| <code>TFLOW</code> | <code>O_NONBLOCK</code> was set, but the flow control mechanism prevented the transport provider from accepting any data at this time. |
| <code>TLOOK</code> | An asynchronous event has occurred on this transport endpoint. |
| <code>TNOTSUPPORT</code> | This function is not supported by the underlying transport provider. |
| <code>TOUTSTATE</code> | The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. |
| <code>TPROTO</code> | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). |

TSYSERR A system error has occurred during execution of this function.

TLI COMPATIBILITY

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```
#include <tiuser.h>
```

Error Description Values

The `t_errno` values that can be set by the XTI interface and cannot be set by the TLI interface are:

TPROTO

TBADADDR

TBADOPT

TLOOK

TOUTSTATE

Notes

Whenever this function fails with `t_error` set to `TFLOW`, `O_NONBLOCK` must have been set.

Option Buffers

The format of the options in an `opt` buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not fix the buffer format.

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`fcntl(2)`, `t_alloc(3N)`, `t_bind(3N)`, `t_error(3N)`, `t_getinfo(3N)`, `t_look(3N)`, `t_open(3N)`, `t_rcvudata(3N)`, `t_rcvuderr(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

| | |
|--------------------|---|
| NAME | t_sndv – send data or expedited data, from one or more non-contiguous buffers, on a connection |
| SYNOPSIS | <pre>#include <xti.h> int t_sndv(intfd, const struct t_iovec * iov, unsigned intiovcnt, intflags);</pre> |
| DESCRIPTION | <p>This function is used to send either normal or expedited data. The argument <i>fd</i> identifies the local transport endpoint over which data should be sent, <i>iov</i> points to an array of buffer address/buffer length pairs. t_sndv() sends data contained in buffers <i>iov0</i> , <i>iov1</i> , through <i>iov</i> [<i>iovcnt-1</i>]. <i>iovcnt</i> contains the number of non-contiguous data buffers which is limited to T_IOV_MAX , an implementation-defined value of at least 16. If the limit is exceeded, the function fails with TBADDDATA.</p> <pre>iov(0).iov_len + . . . + iov(iovcnt-1).iov_len)</pre> <p>Note that the limit on the total number of bytes available in all buffers passed: may be constrained by implementation limits. If no other constraint applies, it will be limited by INT_MAX. In practice, the availability of memory to an application is likely to impose a lower limit on the amount of data that can be sent or received using scatter/gather functions.</p> <p>The argument <i>flags</i> specifies any optional flags described below:</p> <p>T_EXPEDITED If set in <i>flags</i>, the data will be sent as expedited data and will be subject to the interpretations of the transport provider.</p> <p>T_MORE If set in <i>flags</i>, this indicates to the transport provider that the transport service data unit (TSDU) (or expedited transport service data unit – ETSDU) is being sent through multiple t_sndv() calls. Each t_sndv() with the T_MORE flag set indicates that another t_sndv() or t_snd(3N) will follow with more data for the current TSDU (or ETSDU).</p> <p>The end of the TSDU (or ETSDU) is identified by a t_sndv() call with the T_MORE flag not set. Use of T_MORE enables a user to break up large logical data units without losing the boundaries of those units at the other end of the connection. The flag implies nothing about how the data is packaged for transfer below the transport interface. If the transport provider does not support the concept of a TSDU as indicated in the <i>info</i> argument on return from t_open(3N) or t_getinfo(3N), the T_MORE flag is not meaningful and will be ignored if set.</p> |

The sending of a zero-length fragment of a TSDU or ETSDU is only permitted where this is used to indicate the end of a TSDU or ETSDU, that is, when the `T_MORE` flag is not set. Some transport providers also forbid zero-length TSDUs and ETSDUs.

If set in *flags*, requests that the provider transmit all data that it has accumulated but not sent. The request is a local action on the provider and does not affect any similarly named protocol flag (for example, the TCP PUSH flag). This effect of setting this flag is protocol-dependent, and it may be ignored entirely by transport providers which do not support the use of this feature.

The communications provider is free to collect data in a send buffer until it accumulates a sufficient amount for transmission.

By default, `t_sndv()` operates in synchronous mode and may wait if flow control restrictions prevent the data from being accepted by the local transport provider at the time the call is made. However, if `O_NONBLOCK` is set by means of `t_open(3N)` or `fcntl(2)`, `t_sndv()` executes in asynchronous mode, and will fail immediately if there are flow control restrictions. The process can arrange to be informed when the flow control restrictions are cleared via either `t_look(3N)` or the EM interface.

On successful completion, `t_sndv()` returns the number of bytes accepted by the transport provider. Normally this will equal the total number of bytes to be sent, that is,

```
(iov0.iov_len + .. + iov[iovcount-1].iov_len)
```

However, the interface is constrained to send at most `INT_MAX` bytes in a single send. When `t_sndv()` has submitted `INT_MAX` (or lower constrained value, see the note above) bytes to the provider for a single call, this value is returned to the user. However, if `O_NONBLOCK` is set or the function is interrupted by a signal, it is possible that only part of the data has actually been accepted by the communications provider. In this case, `t_sndv()` returns a value that is less than the value of `nbytes`. If `t_sndv()` is interrupted by a signal before it could transfer data to the communications provider, it returns `-1` with `t_errno` set to `TSYSERR` and `errno` set to `EINTR`.

If the number of bytes of data in the *iov* array is zero and sending of zero octets is not supported by the underlying transport service, `t_sndv()` returns `-1` with `t_errno` set to `TBADDATA`.

The size of each TSDU or ETSDU must not exceed the limits of the transport provider as specified by the current values in the TSDU or ETSDU fields in the *info* argument returned by `t_getinfo(3N)`.

RETURN VALUES

The error `TLOOK` is returned for asynchronous events. It is required only for an incoming disconnect event but may be returned for other events.

On successful completion, `t_sndv()` returns the number of bytes accepted by the transport provider. Otherwise, `-1` is returned on failure and `t_errno` is set to indicate the error.

Note that in synchronous mode, if more than `INT_MAX` bytes of data are passed in the *iov* array, only the first `INT_MAX` bytes will be passed to the provider.

If the number of bytes accepted by the communications provider is less than the number of bytes requested, this may either indicate that `O_NONBLOCK` is set and the communications provider is blocked due to flow control, or that `O_NONBLOCK` is clear and the function was interrupted by a signal.

VALID STATES

`T_DATAXFER`, `T_INREL`.

ERRORS

On failure, `t_errno` is set to one of the following:

| | |
|-----------------------|--|
| <code>TBADDATA</code> | Illegal amount of data: |
| <code>TBADF</code> | The specified file descriptor does not refer to a transport endpoint. <ul style="list-style-type: none"> ■ A single send was attempted specifying a TSDU (ETSDU) or fragment TSDU (ETSDU) greater than that specified by the current values of the TSDU or ETSDU fields in the <i>info</i> argument. ■ A send of a zero byte TSDU (ETSDU) or zero byte fragment of a TSDU (ETSDU) is not supported by the provider. ■ Multiple sends were attempted resulting in a TSDU (ETSDU) larger than that specified by the current value of the TSDU or ETSDU fields in the <i>info</i> argument – the ability of an XTI implementation to detect such an error case is implementation-dependent. See <code>WARNINGS</code>, below. ■ <i>iovcount</i> is greater than <code>T_IOV_MAX</code>. |
| <code>TBADFLAG</code> | An invalid flag was specified. |
| <code>TFLOW</code> | <code>O_NONBLOCK</code> was set, but the flow control mechanism prevented the transport provider from accepting any data at this time. |
| <code>TLOOK</code> | An asynchronous event has occurred on this transport endpoint. |

| | |
|-------------|---|
| TNOTSUPPORT | This function is not supported by the underlying transport provider. |
| TOUTSTATE | The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. |
| TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). |
| TSYSERR | A system error has occurred during execution of this function. |

**TLI
COMPATIBILITY
ATTRIBUTES**

In the TLI interface definition, no counterpart of this routine was defined.

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`t_getinfo(3N)`, `t_open(3N)`, `t_rcvv(3N)` `t_rcv(3N)`, `t_snd(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

WARNINGS

It is important to remember that the transport provider treats all users of a transport endpoint as a single user. Therefore if several processes issue concurrent `t_sndv()` or `t_snd(3N)` calls, then the different data may be intermixed.

Multiple sends which exceed the maximum TSDU or ETSDU size may not be discovered by XTI. In this case an implementation-dependent error will result (generated by the transport provider), perhaps on a subsequent XTI call. This error may take the form of a connection abort, a `TSYSERR`, a `TBADDATA` or a `TPROTO` error.

If multiple sends which exceed the maximum TSDU or ETSDU size are detected by XTI, `t_sndv()` fails with `TBADDATA`.

| | |
|--------------------|--|
| NAME | t_sndvudata – send a data unit from one or more noncontiguous buffers |
| SYNOPSIS | <pre>#include <xti.h></pre> <pre>int t_sndvudata(intfd, struct t_unitdata *unitdata, struct t_iovec *iov, unsigned int iovcount);</pre> |
| DESCRIPTION | <p>This function is used in connectionless mode to send a data unit to another transport user. The argument <i>fd</i> identifies the local transport endpoint through which data will be sent, <i>iovcount</i> contains the number of non-contiguous <i>udata</i> buffers and is limited to an implementation-defined value given by <code>T_IOV_MAX</code> which is at least 16, and <i>unitdata</i> points to a <code>t_unitdata</code> structure containing the following members:</p> <pre> struct netbuf addr; struct netbuf opt; struct netbuf udata; </pre> <p>If the limit on <i>iovcount</i> is exceeded, the function fails with <code>TBADDATA</code>.</p> <p>In <i>unitdata</i>, <i>addr</i> specifies the protocol address of the destination user, and <i>opt</i> identifies options that the user wants associated with this request. The <i>udata</i> field is not used. The user may choose not to specify what protocol options are associated with the transfer by setting the <i>len</i> field of <i>opt</i> to zero. In this case, the provider may use default options.</p> <p>The data to be sent is identified by <i>iov</i>[0] through <i>iov</i> [<i>iovcount</i>-1].</p> <p>Note that the limit on the total number of bytes available in all buffers passed:</p> <pre> iov(0).iov_len + . . . + iov(iovcount-1).iov_len </pre> <p>may be constrained by implementation limits. If no other constraint applies, it will be limited by <code>INT_MAX</code>. In practice, the availability of memory to an application is likely to impose a lower limit on the amount of data that can be sent or received using scatter/gather functions.</p> <p>By default, <code>t_sndvudata()</code> operates in synchronous mode and may wait if flow control restrictions prevent the data from being accepted by the local transport provider at the time the call is made. However, if <code>O_NONBLOCK</code> is set by means of <code>t_open(3N)</code> or <code>fcntl(2)</code>, <code>t_sndvudata()</code> executes in asynchronous mode and will fail under such conditions. The process can arrange to be notified of</p> |

the clearance of a flow control restriction by means of either `t_look(3N)` or the EM interface.

If the amount of data specified in `iov0` through `iov [iovcount-1]` exceeds the TSDU size as returned in the `t_sdu` field of the `info` argument of `t_open(3N)` or `t_getinfo(3N)`, or is zero and sending of zero octets is not supported by the underlying transport service, a `TBADDATA` error is generated. If `t_sndvudata()` is called before the destination user has activated its transport endpoint (see `t_bind(3N)`), the data unit may be discarded.

If it is not possible for the transport provider to immediately detect the conditions that cause the errors `TBADADDR` and `TBADOPT`, these errors will alternatively be returned by `t_rcvuderr(3N)`. An application must therefore be prepared to receive these errors in both of these ways.

RETURN VALUES

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and `t_errno` is set to indicate an error.

VALID STATES

`T_IDLE`.

ERRORS

On failure, `t_errno` is set to one of the following:

| | |
|--------------------------|---|
| <code>TBADADDR</code> | The specified protocol address was in an incorrect format or contained illegal information. |
| <code>TBADDATA</code> | Illegal amount of data. <ul style="list-style-type: none"> ■ A single send was attempted specifying a TSDU greater than that specified in the <code>info</code> argument, or a send of a zero byte TSDU is not supported by the provider. ■ <code>iovcount</code> is greater than <code>T_IOV_MAX</code>. |
| <code>TBADF</code> | The specified file descriptor does not refer to a transport endpoint. |
| <code>TBADOPT</code> | The specified options were in an incorrect format or contained illegal information. |
| <code>TFLOW</code> | <code>O_NONBLOCK</code> was set, but the flow control mechanism prevented the transport provider from accepting any data at this time. |
| <code>TLOOK</code> | An asynchronous event has occurred on this transport endpoint. |
| <code>TNOTSUPPORT</code> | This function is not supported by the underlying transport provider. |

TOUTSTATE The communications endpoint referenced by *fd* is not in one of the states in which a call to this function is valid.

TPROTO This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (*t_errno*).

TSYSERR A system error has occurred during execution of this function.

**TLI
COMPATIBILITY
ATTRIBUTES**

In the TLI interface definition, no counterpart of this routine was defined.

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

fcntl(2), **t_alloc(3N)**, **t_open(3N)**, **t_rcvudata(3N)**,
t_rcvvudata(3N) **t_rcvuderr(3N)**, **t_sndudata(3N)**, **attributes(5)**

Transport Interfaces Programming Guide

| | |
|--------------------------|---|
| NAME | t_strerror - produce an error message string |
| SYNOPSIS | <pre>#include <xti.h> const char *t_strerror(int <i>errnum</i>);</pre> |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>The <code>t_strerror()</code> function maps the error number in <code>errnum</code> that corresponds to an XTI error to a language-dependent error message string and returns a pointer to the string. The string pointed to will not be modified by the program, but may be overwritten by a subsequent call to the <code>t_strerror</code> function. The string is not terminated by a newline character. The language for error message strings written by <code>t_strerror()</code> is that of the current locale. If it is English, the error message string describing the value in <code>t_errno</code> may be derived from the comments following the <code>t_errno</code> codes defined in <code><xti.h></code>. If an error code is unknown, and the language is English, <code>t_strerror()</code> returns the string:</p> <pre>"<error>: error unknown"</pre> <p>where <code><error></code> is the error number supplied as input. In other languages, an equivalent text is provided.</p> |
| VALID STATES | ALL - apart from T_UNINIT. |
| RETURN VALUES | The function <code>t_strerror()</code> returns a pointer to the generated message string. |
| TLI COMPATIBILITY | The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below. |
| Interface Header | <p>The XTI interfaces use the header file, <code>xti.h</code>. TLI interfaces should <i>not</i> use this header. They should use the header:</p> <pre>#include <tiuser.h></pre> <p>For more information refer to the <i>Transport Interfaces Programming Guide</i></p> |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

t_errno(3N), **t_error(3N)**, **attributes(5)**

Transport Interfaces Programming Guide

| | | | | | | | |
|----------------------|---|---------|----------|--------|-------|----------|------------------------------|
| NAME | t_sync – synchronize transport library | | | | | | |
| SYNOPSIS | #include <xti.h> int t_sync(int fd); | | | | | | |
| DESCRIPTION | <p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>For the transport endpoint specified by <i>fd</i>, t_sync() synchronizes the data structures managed by the transport library with information from the underlying transport provider. In doing so, it can convert an uninitialized file descriptor (obtained by means of a <code>open(2)</code>, <code>dup(2)</code> or as a result of a <code>fork(2)</code> and <code>exec(2)</code>) to an initialized transport endpoint, assuming that the file descriptor referenced a transport endpoint, by updating and allocating the necessary library data structures. This function also allows two cooperating processes to synchronize their interaction with a transport provider.</p> <p>For example, if a process forks a new process and issues an <code>exec(2)</code>, the new process must issue a t_sync() to build the private library data structure associated with a transport endpoint and to synchronize the data structure with the relevant provider information.</p> <p>It is important to remember that the transport provider treats all users of a transport endpoint as a single user. If multiple processes are using the same endpoint, they should coordinate their activities so as not to violate the state of the transport endpoint. The function t_sync() returns the current state of the transport endpoint to the user, thereby enabling the user to verify the state before taking further action. This coordination is only valid among cooperating processes; it is possible that a process or an incoming event could change the endpoint's state <i>after</i> a t_sync() is issued.</p> <p>If the transport endpoint is undergoing a state transition when t_sync() is called, the function will fail.</p> | | | | | | |
| RETURN VALUES | <p>On successful completion, the state of the transport endpoint is returned. Otherwise, a value of -1 is returned and <code>t_errno</code> is set to indicate an error. The state returned is one of the following:</p> <table border="0"> <tr> <td style="padding-right: 20px;">T_UNBND</td> <td>Unbound.</td> </tr> <tr> <td style="padding-right: 20px;">T_IDLE</td> <td>Idle.</td> </tr> <tr> <td style="padding-right: 20px;">T_OUTCON</td> <td>Outgoing connection pending.</td> </tr> </table> | T_UNBND | Unbound. | T_IDLE | Idle. | T_OUTCON | Outgoing connection pending. |
| T_UNBND | Unbound. | | | | | | |
| T_IDLE | Idle. | | | | | | |
| T_OUTCON | Outgoing connection pending. | | | | | | |

| | |
|------------|---|
| T_INCON | Incoming connection pending. |
| T_DATAXFER | Data transfer. |
| T_OUTREL | Outgoing orderly release (waiting for an orderly release indication). |
| T_INREL | Incoming orderly release (waiting for an orderly release request). |

ERRORS

On failure, `t_errno` is set to one of the following:

| | |
|------------|---|
| TBADF | The specified file descriptor does not refer to a transport endpoint. This error may be returned when the <i>fd</i> has been previously closed or an erroneous number may have been passed to the call. |
| TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). |
| TSTATECHNG | The transport endpoint is undergoing a state change. |
| TSYSERR | A system error has occurred during execution of this function. |

**TLI
COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```
#include <tiuser.h>
```

**Error Description
Values**

The `t_errno` value that can be set by the XTI interface and cannot be set by the TLI interface is:

```
TPROTO
```

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT Level | Safe |

SEE ALSO

`dup(2)`, `exec(2)`, `fork(2)`, `open(2)`, `attributes(5)`

Transport Interfaces Programming Guide

| NAME | t_sysconf – get configurable XTI variables | | | | |
|--------------------------|---|----------------|-----------------|-----------|---------------|
| SYNOPSIS | <pre>#include <xti.h> int t_sysconf(intname);</pre> | | | | |
| DESCRIPTION | <p>The t_sysconf() function provides a method for the application to determine the current value of configurable and implementation-dependent XTI limits or options.</p> <p>The <i>name</i> argument represents the XTI system variable to be queried. The following table lists the minimal set of XTI system variables from <code><xti.h></code> that can be returned by t_sysconf(), and the symbolic constants, defined in <code><xti.h></code> that are the corresponding values used for <i>name</i>.</p> <table border="1" data-bbox="487 823 1383 909"> <thead> <tr> <th>Variable</th> <th>Value of Name</th> </tr> </thead> <tbody> <tr> <td>T_IOV_MAX</td> <td>_SC_T_IOV_MAX</td> </tr> </tbody> </table> | Variable | Value of Name | T_IOV_MAX | _SC_T_IOV_MAX |
| Variable | Value of Name | | | | |
| T_IOV_MAX | _SC_T_IOV_MAX | | | | |
| RETURN VALUES | If <i>name</i> is valid, t_sysconf() returns the value of the requested limit/option, which might be -1, and leaves <code>t_errno</code> unchanged. Otherwise, a value of -1 is returned and <code>t_errno</code> is set to indicate an error. | | | | |
| VALID STATES | All. | | | | |
| ERRORS | On failure, <code>t_errno</code> is set to the following: TBADFLAG <i>name</i> has an invalid value. | | | | |
| TLI COMPATIBILITY | In the TLI interface definition, no counterpart of this routine was defined. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1" data-bbox="487 1404 1383 1491"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | <p>sysconf(3C), t_rcvv(3N), t_rcvvudata(3N), t_sndv(3N), t_sndvudata(3N), attributes(5)</p> <p><i>Transport Interfaces Programming Guide</i></p> | | | | |

| | |
|----------------------|---|
| NAME | ttyname, ttyname_r – find pathname of a terminal |
| SYNOPSIS | <pre>#include <unistd.h> char * ttyname(int fildes); char * ttyname_r(int fildes, char * name, int namelen);</pre> |
| POSIX | <pre>cc [flag ...] file ... -D_POSIX_PTHREAD_SEMANTICS [library ...] int ttyname_r(int fildes, char * name, size_t namesize);</pre> |
| DESCRIPTION | <p>The ttyname() function returns a pointer to a string containing the null-terminated path name of the terminal device associated with file descriptor <i>fildes</i>. The return value may point to static data whose content is overwritten by each call.</p> <p>The ttyname_r() function has the same functionality as ttyname() except that the caller must supply a buffer <i>name</i> with length <i>namelen</i> to store the result; this buffer must be at least <code>_POSIX_PATH_MAX</code> in size (defined in <code><limits.h></code>). The POSIX version (see standards(5)) of ttyname_r() takes a <i>namesize</i> parameter of type <code>size_t</code>.</p> |
| RETURN VALUES | <p>Upon successful completion, ttyname() and ttyname_r() return a pointer to a string. Otherwise, a null pointer is returned and <code>errno</code> is set to indicate the error.</p> <p>The POSIX ttyname_r() returns zero if successful, or the error number upon failure.</p> |
| ERRORS | <p>The ttyname_r() function will fail if:</p> <p>ERANGE The size of the buffer is smaller than the result to be returned.</p> <p>The ttyname() function may fail if:</p> <p>EBADF The <i>fildes</i> argument is not a valid file descriptor.</p> <p>ENOTTY The <i>fildes</i> argument does not refer to a terminal device.</p> |
| FILES | <pre>/dev/* device file</pre> |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | See NOTES below. |

SEE ALSO

Intro(3), **gettext(3C)**, **setlocale(3C)**, **attributes(5)**, **standards(5)**

NOTES

When compiling multithreaded programs, see **Intro(3)**, *Notes On Multithreaded Applications*.

If the application is linked with `-lintl`, then messages printed from this function are in the native language specified by the `LC_MESSAGES` locale category; see **setlocale(3C)**.

The return value points to static data whose content is overwritten by each call.

The **ttyname()** is Unsafe in multithreaded applications. The **ttyname_r()** function is MT-Safe, and should be used instead.

Solaris 2.4 and earlier releases provided definitions of the **ttyname_r()** interface as specified in POSIX.1c Draft 6. The final POSIX.1c standard changed the interface as described above. Support for the Draft 6 interface is provided for compatibility only and may not be supported in future releases. New applications and libraries should use the POSIX standard interface.

NAME | ttyslot – find the slot in the utmp file of the current user

SYNOPSIS | #include <stdlib.h>
 | int ttyslot(void);

DESCRIPTION | The **ttyslot()** function returns the index of the current user's entry in the /var/adm/utmp file. The returned index is accomplished by scanning files in /dev for the name of the terminal associated with the standard input, the standard output, or the standard error output (0, 1, or 2).

RETURN VALUES | Upon successful completion, **ttyslot()** returns the index of the current user's entry in the /var/adm/utmp file. If an error was encountered while searching for the terminal name or if none of the above file descriptors are associated with a terminal device, -1 is returned.

FILES | /var/adm/utmp user access and accounting information

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO | **getutent(3C)**, **ttyname(3C)**, **utmp(4)**, **attributes(5)**

| | | | | | | | | | | | |
|--------------------------|---|-------|---|-------|--|-----------|--|--------|---|---------|--|
| NAME | t_unbind – disable a transport endpoint | | | | | | | | | | |
| SYNOPSIS | <pre>#include <xti.h> int t_unbind(int fd);</pre> | | | | | | | | | | |
| DESCRIPTION | <p>The This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <code>tiuser.h</code> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>t_unbind() function disables the transport endpoint specified by <i>fd</i> which was previously bound by t_bind(3N). On completion of this call, no further data or events destined for this transport endpoint will be accepted by the transport provider. An endpoint which is disabled by using t_unbind() can be enabled by a subsequent call to t_bind(3N).</p> | | | | | | | | | | |
| RETURN VALUES | Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and <code>t_errno</code> is set to indicate an error. | | | | | | | | | | |
| VALID STATES | T_IDLE. | | | | | | | | | | |
| ERRORS | <p>On failure, <code>t_errno</code> is set to one of the following:</p> <table border="0"> <tr> <td style="padding-right: 20px;">TBADF</td> <td>The specified file descriptor does not refer to a transport endpoint.</td> </tr> <tr> <td style="padding-right: 20px;">TLOOK</td> <td>An asynchronous event has occurred on this transport endpoint.</td> </tr> <tr> <td style="padding-right: 20px;">TOUTSTATE</td> <td>The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid.</td> </tr> <tr> <td style="padding-right: 20px;">TPROTO</td> <td>This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>).</td> </tr> <tr> <td style="padding-right: 20px;">TSYSERR</td> <td>A system error has occurred during execution of this function.</td> </tr> </table> | TBADF | The specified file descriptor does not refer to a transport endpoint. | TLOOK | An asynchronous event has occurred on this transport endpoint. | TOUTSTATE | The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. | TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). | TSYSERR | A system error has occurred during execution of this function. |
| TBADF | The specified file descriptor does not refer to a transport endpoint. | | | | | | | | | | |
| TLOOK | An asynchronous event has occurred on this transport endpoint. | | | | | | | | | | |
| TOUTSTATE | The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid. | | | | | | | | | | |
| TPROTO | This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>). | | | | | | | | | | |
| TSYSERR | A system error has occurred during execution of this function. | | | | | | | | | | |
| TLI COMPATIBILITY | The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below. | | | | | | | | | | |

Interface Header

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```
#include <tiuser.h>
```

Error Description Values

The `t_errno` value that can be set by the XTI interface and cannot be set by the TLI interface is:

```
TPROTO
```

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`t_bind(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

| | |
|----------------------|--|
| NAME | typeahead – check for type-ahead characters |
| SYNOPSIS | <pre>#include <curses.h> int typeahead(int fd);</pre> |
| PARAMETERS | <p>fd Is the file descriptor that is used to check for type-ahead characters.</p> |
| DESCRIPTION | <p>The typeahead() function specifies the file descriptor (<i>fd</i>) to use to check for type-ahead characters (characters typed by the user but not yet processed by X/Open Curses).</p> <p>X/Open Curses checks for type-ahead characters periodically while updating the screen. If characters are found, the current update is postponed until the next refresh(3XC) or doupdate(3XC). This speeds up response to commands that have been typed ahead. Normally, the input file pointer passed to newterm(3XC), or <code>stdin</code> in the case of initscr(3XC), is used for type-ahead checking.</p> <p>If <i>fd</i> is -1, no type-ahead checking is done.</p> |
| RETURN VALUES | On success, the typeahead() function returns OK. Otherwise, it returns ERR. |
| ERRORS | None. |
| SEE ALSO | doupdate(3XC) , getch(3XC) , initscr(3XC) |

| | |
|----------------------|---|
| NAME | ualarm – schedule signal after interval in microseconds |
| SYNOPSIS | <pre>#include <unistd.h> useconds_t ualarm(useconds_t useconds, useconds_t interval);</pre> |
| DESCRIPTION | <p>The ualarm() function causes the SIGALRM signal to be generated for the calling process after the number of real-time microseconds specified by the <i>useconds</i> argument has elapsed. When the <i>interval</i> argument is non-zero, repeated timeout notification occurs with a period in microseconds specified by the <i>interval</i> argument. If the notification signal, SIGALRM, is not caught or ignored, the calling process is terminated.</p> <p>Because of scheduling delays, resumption of execution when the signal is caught may be delayed an arbitrary amount of time.</p> <p>Interactions between ualarm() and either alarm(2) or sleep(3C) are unspecified.</p> |
| RETURN VALUES | The ualarm() function returns the number of microseconds remaining from the previous ualarm() call. If no timeouts are pending or if ualarm() has not previously been called, ualarm() returns 0. |
| ERRORS | No errors are defined. |
| USAGE | The ualarm() function is a simplified interface to setitimer(2) , and uses the ITIMER_REAL interval timer. |
| SEE ALSO | alarm(2) , setitimer(2) , sighold(3C) , signal(3C) , sleep(3C) , usleep(3C) |

| NAME | ungetc – push byte back into input stream | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <stdio.h> int ungetc(int c, FILE *stream);</pre> | | | | |
| DESCRIPTION | <p>The ungetc() function pushes the byte specified by <i>c</i> (converted to an unsigned char) back onto the input stream pointed to by <i>stream</i>. The pushed-back bytes will be returned by subsequent reads on that stream in the reverse order of their pushing. A successful intervening call (with the stream pointed to by <i>stream</i>) to a file-positioning function (fseek(3S), fsetpos(3S) or rewind(3S)) discards any pushed-back bytes for the stream. The external storage corresponding to the stream is unchanged.</p> <p>Four bytes of push-back are guaranteed. If ungetc() is called too many times on the same stream without an intervening read or file-positioning operation on that stream, the operation may fail.</p> <p>If the value of <i>c</i> equals that of the macro EOF, the operation fails and the input stream is unchanged.</p> <p>A successful call to ungetc() clears the end-of-file indicator for the stream. The value of the file-position indicator for the stream after reading or discarding all pushed-back bytes will be the same as it was before the bytes were pushed back. The file-position indicator is decremented by each successful call to ungetc(); if its value was 0 before a call, its value is indeterminate after the call.</p> | | | | |
| RETURN VALUES | Upon successful completion, ungetc() returns the byte pushed back after conversion. Otherwise it returns EOF . | | | | |
| ERRORS | No errors are defined. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | read(2) , intro(3) , fseek(3S) , fsetpos(3S) , getc(3S) , setbuf(3S) , stdio(3S) , attributes(5) | | | | |

| | | | | | |
|----------------------|---|------------------|---|-------------------|--|
| NAME | ungetch, unget_wch – push character back onto the input queue | | | | |
| SYNOPSIS | <pre>#include <curses.h> int ungetch(int <i>ch</i>); int unget_wch(const wchar_t <i>wch</i>);</pre> | | | | |
| PARAMETERS | <table><tr><td><i>ch</i></td><td>Is the single byte character to be put back in the input queue for the next call to getch(3XC) .</td></tr><tr><td><i>wch</i></td><td>Is the wide character to be put back in the input queue for the next call to get_wch(3XC) .</td></tr></table> | <i>ch</i> | Is the single byte character to be put back in the input queue for the next call to getch(3XC) . | <i>wch</i> | Is the wide character to be put back in the input queue for the next call to get_wch(3XC) . |
| <i>ch</i> | Is the single byte character to be put back in the input queue for the next call to getch(3XC) . | | | | |
| <i>wch</i> | Is the wide character to be put back in the input queue for the next call to get_wch(3XC) . | | | | |
| DESCRIPTION | <p>The ungetch() function pushes <i>ch</i> back onto the input queue until the next call to getch() .</p> <p>The unget_wch() function is similar to ungetch() except that <i>ch</i> can be of type <code>wchar_t</code> .</p> | | | | |
| RETURN VALUES | On success, these functions return <code>OK</code> . Otherwise, they return <code>ERR</code> . | | | | |
| ERRORS | None. | | | | |
| SEE ALSO | get_wch(3XC) , getch(3XC) | | | | |

| NAME | ungetwc – push wide-character code back into input stream | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <stdio.h> #include <wchar.h> wint_t ungetwc(wint_t wc, FILE *stream);</pre> | | | | |
| DESCRIPTION | <p>The ungetwc() function pushes the character corresponding to the wide character code specified by <i>wc</i> back onto the input stream pointed to by <i>stream</i>. The pushed-back characters will be returned by subsequent reads on that stream in the reverse order of their pushing. A successful intervening call (with the stream pointed to by <i>stream</i>) to a file-positioning function (fseek(3S), fsetpos(3S) or rewind(3S)) discards any pushed-back characters for the stream. The external storage corresponding to the stream is unchanged.</p> <p>One character of push-back is guaranteed. If ungetwc() is called too many times on the same stream without an intervening read or file-positioning operation on that stream, the operation may fail.</p> <p>If the value of <i>wc</i> equals that of the macro WEOF, the operation fails and the input stream is unchanged.</p> <p>A successful call to ungetwc() clears the end-of-file indicator for the stream. The value of the file-position indicator for the stream after reading or discarding all pushed-back characters will be the same as it was before the characters were pushed back. The file-position indicator is decremented (by one or more) by each successful call to ungetwc(); if its value was 0 before a call, its value is indeterminate after the call.</p> | | | | |
| RETURN VALUES | Upon successful completion, ungetwc() returns the wide-character code corresponding to the pushed-back character. Otherwise it returns WEOF . | | | | |
| ERRORS | <p>The ungetwc() function may fail if:</p> <p>EILSEQ An invalid character sequence is detected, or a wide-character code does not correspond to a valid character.</p> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | read(2) , fseek(3S) , fsetpos(3S) , rewind(3S) , setbuf(3S) , attributes(5) | | | | |

NAME unlockpt – unlock a pseudo-terminal master/slave pair

SYNOPSIS #include <stdlib.h>

```
int unlockpt(int fildes);
```

DESCRIPTION The **unlockpt()** function unlocks the slave pseudo-terminal device associated with the master to which *fildes* refers.

Portable applications must call **unlockpt()** before opening the slave side of a pseudo-terminal device.

RETURN VALUES Upon successful completion, **unlockpt()** returns 0. Otherwise, it returns -1 and sets `errno` to indicate the error.

ERRORS The **unlockpt()** function may fail if:

EBADF The *fildes* argument is not a file descriptor open for writing.

EINVAL The *fildes* argument is not associated with a master pseudo-terminal device.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO **open(2)**, **grantpt(3C)**, **ptsname(3C)**, **attributes(5)**

STREAMS Programming Guide

| | |
|----------------------|---|
| NAME | use_env – set values of lines and columns |
| SYNOPSIS | <pre>#include <curses.h> void use_env(char <i>bool</i>);</pre> |
| PARAMETERS | <p><i>bool</i> Is a Boolean expression.</p> |
| DESCRIPTION | <p>The use_env() function takes the values for lines and columns from the terminfo database (if <i>bool</i> is FALSE) or from environmental variables LINES and COLUMNS (if <i>bool</i> is TRUE). If no environmental variables have been set, the window size is used. This function must be set before initscr(3XC), newterm(3XC), or setupterm(3XC) is called. The default action is TRUE.</p> |
| RETURN VALUES | The use_env() function does not return a value. |
| ERRORS | None. |
| SEE ALSO | del_curterm(3XC) , initscr(3XC) |

| NAME | usleep – suspend execution for interval in microseconds | | | | |
|----------------------|--|----------------|-----------------|----------|--------|
| SYNOPSIS | #include <unistd.h> int usleep (useconds_t <i>useconds</i>); | | | | |
| DESCRIPTION | <p>The usleep() function suspends the current process from execution for the number of microseconds specified by the <i>useconds</i> argument. (A microsecond is .000001 seconds.) Because of other activity, or because of the time spent in processing the call, the actual suspension time may be longer than the amount of time specified.</p> <p>The <i>useconds</i> argument must be less than 1,000,000. If the value of <i>useconds</i> is 0, then the call has no effect.</p> <p>The usleep() function uses the process's real-time interval timer to indicate to the system when the process should be woken up.</p> <p>There is one real-time interval timer for each process. The usleep() function will not interfere with a previous setting of this timer. If the process has set this timer prior to calling usleep(), and if the time specified by <i>useconds</i> equals or exceeds the interval timer's prior setting, the process will be woken up shortly before the timer was set to expire.</p> <p>Interactions between usleep() and either alarm(2) or sleep(3C) are unspecified.</p> | | | | |
| RETURN VALUES | On successful completion, usleep() returns 0. Otherwise, it returns -1 and sets <i>errno</i> to indicate the error. | | | | |
| ERRORS | The usleep() function may fail if: EINVAL The time interval specified 1,000,000 or more microseconds. | | | | |
| USAGE | The usleep() function is included for its historical usage. The setitimer(2) function is preferred over this function. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | alarm(2) , poll(2) , setitimer(2) , sigaction(2) , sigprocmask(2) , select(3C) , sleep(3C) , ualarm(3C) | | | | |

| NAME | vfwprintf, vwprintf, vswprintf – wide-character formatted output of a stdarg argument list | | | | |
|----------------------|---|----------------|-----------------|----------|-------------------------|
| SYNOPSIS | <pre>#include <stdarg.h> #include <stdio.h> #include <wchar.h> int vwprintf(const wchar_t * format, va_list arg); int vfwprintf(FILE * stream, const wchar_t * format, va_list arg); int vswprintf(wchar_t * s, size_t n, const wchar_t * format, va_list arg);</pre> | | | | |
| DESCRIPTION | <p>The vwprintf() , vfwprintf() , and vswprintf() functions are the same as wprintf() , fwprintf() , and swprintf() respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by <stdarg.h> . See stdarg(5) .</p> <p>These functions do not invoke the va_end() macro. However, as these functions do invoke the va_arg() macro, the value of <i>ap</i> after the return is indeterminate.</p> | | | | |
| RETURN VALUES | Refer to fwprintf(3S) . | | | | |
| ERRORS | Refer to fwprintf(3S) . | | | | |
| USAGE | Applications using these functions should call va_end(ap) afterwards to clean up. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe with exceptions</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe with exceptions |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe with exceptions | | | | |
| SEE ALSO | fwprintf(3S) , setlocale(3C) , attributes(5) , stdarg(5) | | | | |
| NOTES | The vwprintf() , vfwprintf() , and vswprintf() functions can be used safely in multithreaded applications, as long as setlocale(3C) is not being called to change the locale. | | | | |

| | |
|----------------------|--|
| NAME | vidattr, vid_attr, vidputs, vid_puts – display string with video attributes |
| SYNOPSIS | <pre>#include <term.h> int vidattr(chtype attr); int vid_attr(attr_t attr, short color_pair, void * opt); int vidputs(chtype attr, int (* putfunc)(int)); int vid_puts(attr_t attr, short color_pair, void * opt, int (* putfunc)(int));</pre> |
| PARAMETERS | <p>attr Is the rendition of the foreground window.</p> <p>color_pair Is a color pair.</p> <p>opt Is reserved for future use. Currently, this must be a null pointer.</p> <p>putfunc Is a user-supplied output function.</p> <p>putwfunc Is a user-supplied output function.</p> |
| DESCRIPTION | <p>These functions change the terminal's attributes.</p> <p>The vidattr() function sends a request to the terminal to display subsequent characters with the rendition specified by <i>attr</i> . It uses the putchar(3S) function to display the character. The vid_attr() function is similar to the vidattr() function except that it accepts the rendition as a <i>attr_t</i> object. This lets you use the attribute constants that begin with <i>WA_</i> .</p> <p>The vidputs() and vid_puts() functions are similar to the vidattr() and vid_attr() functions, respectively, except that the user-supplied <i>putfunc</i> function is used instead of putchar() . The output of the user-supplied function is ignored by vidputs() and vid_puts() functions.</p> |
| RETURN VALUES | On success, these functions return <i>OK</i> . Otherwise, they return <i>ERR</i> . |
| ERRORS | None. |
| SEE ALSO | doupdate(3XC) , is_linetouched(3XC) , putchar(3S) , tigetflag(3XC) |

| | |
|----------------------|---|
| NAME | vlfmt – display error message in standard format and pass to logging and monitoring services |
| SYNOPSIS | <pre>#include <pfmt.h> #include <stdarg.h> int vlfmt(FILE *stream, long flag, const char *format, va_list ap);</pre> |
| DESCRIPTION | <p>The vlfmt() function is identical to lfmt(3C), except that it is called with an argument list as defined by <code><stdarg.h></code>.</p> <p>The <code><stdarg.h></code> header defines the type <code>va_list</code> and a set of macros for advancing through a list of arguments whose number and types may vary. The <i>ap</i> argument is of type <code>va_list</code>. This argument is used with the <code><stdarg.h></code> macros va_start(), va_arg(), and va_end(). See stdarg(5). The example in the EXAMPLES section below demonstrates their use with vlfmt().</p> |
| RETURN VALUES | <p>Upon successful completion, vlfmt() returns the number of bytes transmitted. Otherwise, <code>-1</code> is returned if there was a write error to <i>stream</i>, or <code>-2</code> is returned if unable to log and/or display at console.</p> |
| EXAMPLES | <p>EXAMPLE 1 Use of vlfmt() to write an errlog() routine.</p> <p>The following example demonstrates how vlfmt() could be used to write an errlog() routine. The va_alist() macro is used as the parameter list in a function definition. The <code>va_start(ap, ...)</code> call, where <i>ap</i> is of type <code>va_list</code>, must be invoked before any attempt to traverse and access unnamed arguments. Calls to <code>va_arg(ap, atype)</code> traverse the argument list. Each execution of va_arg() expands to an expression with the value and type of the next argument in the list <i>ap</i>, which is the same object initialized by va_start(). The <i>atype</i> argument is the type that the returned argument is expected to be. The <code>va_end(ap)</code> macro must be invoked when all desired arguments have been accessed. The argument list in <i>ap</i> can be traversed again if va_start() is called again after va_end(). In the example below, va_arg() is executed first to retrieve the format string passed to errlog(). The remaining errlog() arguments (<i>arg1</i>, <i>arg2</i>, ...) are passed to vlfmt() in the argument <i>ap</i>.</p> <pre>#include <pfmt.h> #include <stdarg.h> /* * errlog should be called like * errlog(log_info, format, arg1, ...); */ void errlog(long log_info, ...) { va_list ap; char *format; va_start(ap,); format = va_arg(ap, char *); (void) vlfmt(stderr, log_info MM_ERROR, format, ap); }</pre> |

```
        va_end(ap);  
        (void) abort();  
    }
```

USAGE

Since **vlfmt()** uses **gettext(3C)**, it is recommended that **vlfmt()** not be used.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

gettext(3C), **lfmt(3C)**, **attributes(5)**, **stdarg(5)**

| | |
|----------------------|---|
| NAME | volmgt_acquire – reserve removable media device |
| SYNOPSIS | <pre>cc [flag ...] file ... -lvolmgt [library ...] #include <sys/types.h> #include <volmgt.h> int volmgt_acquire(char *dev, char *id, int ovr, char **err, pid_t *pidp);</pre> |
| DESCRIPTION | <p>The volmgt_acquire() routine reserves the removable media device specified as <i>dev</i>. volmgt_acquire() operates in two different modes, depending on whether or not Volume Management is running. See void(1M).</p> <p>If Volume Management <i>is</i> running, volmgt_acquire() attempts to reserve the removable media device specified as <i>dev</i>. Specify <i>dev</i> as <i>either</i> a symbolic device name (for example, <code>floppy0</code>) or a physical device pathname (for example, <code>/vol/dsk/unnamed_floppy</code>).</p> <p>If Volume Management <i>is not</i> running, volmgt_acquire() requires callers to specify a physical device pathname for <i>dev</i>. Specifying <i>dev</i> as a symbolic device name is <i>not</i> acceptable. In this mode, volmgt_acquire() relies entirely on the major and minor numbers of the device to determine whether or not the device is reserved.</p> <p>If <i>dev</i> is free, volmgt_acquire() updates the internal device reservation database with the caller's process id (<i>pid</i>) and the specified <i>id</i> string.</p> <p>If <i>dev</i> is reserved by another process, the reservation attempt fails and <code>volmgt_acquire()</code>:</p> <ul style="list-style-type: none"> ■ sets <code>errno</code> to <code>EBUSY</code> ■ fills the caller's <i>id</i> value in the array pointed to by <i>err</i> ■ fills in the <i>pid</i> to which the pointer <i>pidp</i> points with the <i>pid</i> of the process which holds the reservation, if the supplied <i>pidp</i> is non-zero <p>If the override <i>ovr</i> is non-zero, the call overrides the device reservation.</p> |
| RETURN VALUES | <p>Upon successful completion, volmgt_acquire() returns a non-zero value.</p> <p>Upon failure, volmgt_acquire() returns 0. If the return value is 0, and <code>errno</code> is set to <code>EBUSY</code>, the address pointed to by <i>err</i> contains the string that was specified as <i>id</i> (when the device was reserved by the process holding the reservation).</p> |
| ERRORS | <p>The volmgt_acquire() routine fails if one or more of the following are true:</p> <p>EINVAL One of the specified arguments is invalid or missing.</p> <p>EBUSY <i>dev</i> is already reserved by another process (and <i>ovr</i> was not set to a non-zero value)</p> |

EXAMPLES**EXAMPLE 1** A sample of the **volmgt_acquire()** routine.

In the following example, Volume Management is running and the first floppy drive is reserved, accessed and released.

```
#include <volmgt.h>
char *errp;

if (!volmgt_acquire("floppy0", "FileMgr", 0, NULL,
    &errp, NULL)) {
    /* handle error case */
    ...
}
/* floppy acquired - now access it */
if (!volmgt_release("floppy0")) {
    /* handle error case */
    ...
}
}
```

EXAMPLE 2 How callers can override a lock on another process using **volmgt_acquire()**.

The following example shows how callers can override a lock on another process using **volmgt_acquire()**.

```
char *errp, buf[20];
int override = 0;
pid_t pid;

if (!volmgt_acquire("floppy0", "FileMgr", 0, &errp,
    &pid)) {
    if (errno == EBUSY) {
        (void) printf("override %s (pid=%ld)?\n",
            errp, pid); {
        (void) fgets(buf, 20, stdin);
        if (buf[0] == 'y') {
            override++;
        }
    } else {
        /* handle other errors */
        ...
    }
}
if (override) {
    if (!volmgt_acquire("floppy0", "FileMgr", 1,
        &errp, NULL)) {
        /* really give up this time! */
        ...
    }
}
}
```

ATTRIBUTESSee **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO `vold(1M)`, `free(3C)`, `malloc(3C)`, `volmgt_release(3X)`, `attributes(5)`

NOTES When returning a string through *err*, `volmgt_acquire()` allocates a memory area using `malloc(3C)`. Use `free(3C)` to release the memory area when no longer needed.

The *ovr* argument is intended to allow callers to override the current device reservation. It is assumed that the calling application has determined that the current reservation can safely be cleared. See `EXAMPLES`.

NAME | volmgt_check – have Volume Management check for media

SYNOPSIS | `cc [flag ...] file ... -lvolmgt [library ...]`
`#include <volmgt.h>`
`int volmgt_check(char *pathname);`

DESCRIPTION | This routine asks Volume Management to check the specified *pathname* and determine if new media has been inserted in that drive.

If a null pointer is passed in, then Volume Management will check each device it is managing that can be checked.

If new media is found, **volmgt_check()** tells Volume Management to initiate any "actions" specified in `/etc/vold.conf` (see **vold.conf(4)**).

RETURN VALUES | This routine returns 0 if no media was found, and a non-zero value if any media was found.

ERRORS | This routine can fail, returning 0, if a **stat(2)** or **open(2)** of the supplied *pathname* fails, or if any of the following is true:

ENXIO | Volume Management is not running.

EINTR | An interrupt signal was detected while checking for media.

EXAMPLES | **EXAMPLE 1** Checking if any new media is inserted.

To check if any drive managed by Volume Management has any new media inserted in it:

```
if (volmgt_check(NULL)) {
    (void) printf("Volume Management found media\n");
}
```

This would also request Volume Management to take whatever action was specified in `/etc/vold.conf` for any media found.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **cc(1B)**, **volcheck(1)**, **vold(1M)**, **open(2)**, **stat(2)**, **volmgt_inuse(3X)**, **volmgt_running(3X)**, **vold.conf(4)**, **attributes(5)**, **volfs(7FS)**

NOTES

Volume Management must be running for this routine to work.

Since **volmgt_check()** returns 0 for two different cases (both when no media is found, and when an error occurs), it is up to the user to check *errno* to differentiate the two, and to ensure that Volume Management is running.

| NAME | volmgt_feature_enabled – check whether specific Volume Management features are enabled | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag ...] file ... -l volmgt [library ...] #include <volmgt.h> int volmgt_feature_enabled(char *feat_str);</pre> | | | | |
| DESCRIPTION | <p>The volmgt_feature_enabled() routine checks whether specific Volume Management features are enabled. volmgt_feature_enabled() checks for the Volume Management features passed in to it by the <i>feat_str</i> parameter.</p> <p>Currently, the only supported feature string that volmgt_feature_enabled() checks for is floppy-summit-interfaces. The floppy-summit-interfaces feature string checks for the presence of the libvolmgt routines volmgt_acquire() and volmgt_release().</p> <p>The list of features that volmgt_feature_enabled() checks for is expected to expand in the future.</p> | | | | |
| RETURN VALUES | 0 is returned if the specified feature is not currently available. A non-zero value indicates that the specified feature is currently available. | | | | |
| EXAMPLES | <p>EXAMPLE 1 A sample of the volmgt_feature_enabled() function.</p> <p>In the following example, volmgt_feature_enabled() checks whether the floppy-summit-interfaces feature is enabled.</p> <pre>if (volmgt_feature_enabled("floppy-summit-interfaces")) { (void) printf("Media Sharing Routines ARE present\n"); } else { (void) printf("Media Sharing Routines are NOT present\n"); }</pre> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | volmgt_acquire(3X) , volmgt_release(3X) , attributes(5) | | | | |

| NAME | volmgt_inuse – check whether or not Volume Management is managing a <i>pathname</i> | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lvolmgt [<i>library</i> ...] #include <volmgt.h> int volmgt_inuse(char *<i>pathname</i>);</pre> | | | | |
| DESCRIPTION | volmgt_inuse() checks whether Volume Management is managing the specified <i>pathname</i> . | | | | |
| RETURN VALUES | A non-zero value is returned if Volume Management is managing the specified <i>pathname</i> , otherwise 0 is returned. | | | | |
| ERRORS | <p>This routine can fail, returning 0, if a stat(2) of the supplied <i>pathname</i> or an open(2) of <code>/dev/volctl</code> fails, or if any of the following is true:</p> <p>ENXIO Volume Management is not running.</p> <p>EINTR An interrupt signal was detected while checking for the supplied <i>pathname</i> for use.</p> | | | | |
| EXAMPLES | <p>EXAMPLE 1 A sample of the volmgt_inuse() function.</p> <p>To see if Volume Management is managing the first floppy disk:</p> <pre>if (volmgt_inuse("/dev/rdiskette0") != 0) { (void) printf("volmgt is managing diskette 0\n"); } else { (void) printf("volmgt is NOT managing diskette 0\n"); }</pre> | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | cc(1B) , vold(1M) , open(2) , stat(2) , errno(3C) , volmgt_check(3X) , volmgt_running(3X) , attributes(5) , volfs(7FS) | | | | |
| NOTES | <p>This routine requires Volume Management to be running.</p> <p>Since volmgt_inuse() returns 0 for two different cases (both when a volume is not in use, and when an error occurs), it is up to the user to check errno to differentiate the two, and to ensure that Volume Management is running.</p> | | | | |

| NAME | volmgt_ownspath – check Volume Management name space for path | | | | | | |
|----------------------|---|----------------|-----------------|----------|------|------------------|--------|
| SYNOPSIS | <pre>cc [flag ...] file ...-lvolgmt [library ...] #include <volmgt.h> int volmgt_ownspath(char *path);</pre> | | | | | | |
| DESCRIPTION | volmgt_ownspath() checks to see if a given <i>path</i> is contained in the Volume Management name space. This is achieved by comparing the beginning of the supplied path name with the output from volmgt_root(3X) | | | | | | |
| PARAMETERS | path A string containing the path. | | | | | | |
| RETURN VALUES | <p>non-zero The <i>path</i> is owned by Volume Management.</p> <p>0 volgmt() does not have <i>path</i> in its name space, or Volume Management is not running.</p> | | | | | | |
| EXAMPLES | <p>EXAMPLE 1 Using volmgt_ownspath()</p> <p>The following example first checks if volgmt() is running, then checks the Volume Management name space for <i>path</i>, and then returns the <i>id</i> for the piece of media.</p> <pre>char *path; ... if (volmgt_running()) { if (volmgt_ownspath(path)) { (void) printf("id of %s is %lld\n", path, media_getid(path)); } }</pre> | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT Level</td> <td>Safe</td> </tr> <tr> <td>Commitment Level</td> <td>Public</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT Level | Safe | Commitment Level | Public |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| MT Level | Safe | | | | | | |
| Commitment Level | Public | | | | | | |

volmgt_ownspath(3X)

Miscellaneous Library Functions

SEE ALSO | `volmgt_root(3X)`, `volmgt_running(3X)` `attributes(5)`

| | |
|----------------------|--|
| NAME | volmgt_release – release removable media device reservation |
| SYNOPSIS | <pre>cc [flag ...] file ... -lvolmgt [library ...] #include <volmgt.h> int volmgt_release(char *dev);</pre> |
| DESCRIPTION | <p>The volmgt_release() routine releases the removable media device reservation specified as <i>dev</i>. See volmgt_acquire(3X) for a description of <i>dev</i>.</p> <p>If <i>dev</i> is reserved by the caller, volmgt_release() updates the internal device reservation database to indicate that the device is no longer reserved. If the requested device is reserved by another process, the release attempt fails and <code>errno</code> is set to 0.</p> |
| RETURN VALUES | <p>Upon successful completion, <code>volmgt_release</code> returns a non-zero value. Upon failure, 0 is returned.</p> |
| ERRORS | <p>On failure, volmgt_release() returns 0, and sets <code>errno</code> for one of the following conditions:</p> <p>EINVAL <i>dev</i> was invalid or missing.</p> <p>EBUSY <i>dev</i> was not reserved by the caller.</p> |
| EXAMPLES | <p>EXAMPLE 1 A sample of the volmgt_release() routine.</p> <p>In the following example, Volume Management is running, and the first floppy drive is reserved, accessed and released.</p> <pre>#include <volmgt.h> char *errp; if (!volmgt_acquire("floppy0", "FileMgr", 0, &errp, NULL)) { /* handle error case */ ... } /* floppy acquired - now access it */ if (!volmgt_release("floppy0")) { /* handle error case */ ... }</pre> |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: |

volmgt_release(3X)

Miscellaneous Library Functions

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|---------------------|-----------------|
| MT-Level | MT-Safe |
| Interface Stability | Stable |

SEE ALSO

`vold(1M)`, `volmgt_acquire(3X)`, `attributes(5)`

| NAME | volmgt_root – return the Volume Management root directory | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lvolmgt [library ...] #include <volmgt.h> char *volmgt_root(void);</pre> | | | | |
| DESCRIPTION | volmgt_root() returns the current Volume Management root directory, which by default is /vol but can be configured to be in a different location. | | | | |
| RETURN VALUES | A pointer to a static string containing the root directory for Volume Management is returned. | | | | |
| ERRORS | This routine may fail if an open() of /dev/volctl fails. If this occurs a pointer to the default Volume Management root directory is returned. | | | | |
| EXAMPLES | <p>EXAMPLE 1 Finding the Volume Management root directory.</p> <p>To find out where the Volume Management root directory is:</p> <pre>if ((path = volmgt_root()) != NULL) { (void) printf("Volume Management root dir=%s\n", path); } else { (void) printf("can't find Volume Management root dir\n"); }</pre> | | | | |
| FILES | /vol Default location for the Volume Management root directory | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | cc(1B), vold(1M), open(2), volmgt_check(3X), volmgt_inuse(3X), volmgt_running (3X), attributes(5), volfs(7FS) | | | | |
| NOTES | This routine will return the default root directory location even when Volume Management is not running. | | | | |

| NAME | volmgt_running – return whether or not Volume Management is running | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>cc [flag ...] file ... -lvolmgt [library ...] #include <volmgt.h> int volmgt_running(void);</pre> | | | | |
| DESCRIPTION | volmgt_running() tells whether or not Volume Management is running. | | | | |
| RETURN VALUES | A non-zero value is returned if Volume Management is running, else 0 is returned. | | | | |
| ERRORS | <p>volmgt_running() will fail, returning 0, if a stat(2) or open(2) of /dev/volctl fails, or if any of the following is true:</p> <p>ENXIO Volume Management is not running.</p> <p>EINTR An interrupt signal was detected while checking to see if Volume Management was running.</p> | | | | |
| EXAMPLES | <p>EXAMPLE 1 A Sample program of volmgt_running().</p> <p>To see if Volume Management is running:</p> <pre>if (volmgt_running() != 0) { (void) printf("Volume Management is running\n"); } else { (void) printf("Volume Management is NOT running\n"); }</pre> | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">MT-Level</td> <td style="text-align: center;">MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | cc(1B) , vold(1M) , open(2) , stat(2) , volmgt_check(3X) , volmgt_inuse(3X) , attributes(5) , volfs(7FS) | | | | |
| NOTES | Volume Management must be running for many of the Volume Management library routines to work. | | | | |

| | |
|----------------------|--|
| NAME | volmgt_symname, volmgt_symdev – convert between Volume Management symbolic names, and the devices that correspond to them |
| SYNOPSIS | <pre>cc [<i>flag</i> ...] <i>file</i> ... -lvolmgt [<i>library</i> ...] #include <volmgt.h> char * volmgt_symname(char * <i>pathname</i>); char * volmgt_symdev(char * <i>symname</i>);</pre> |
| DESCRIPTION | <p>These two routines compliment each other, translating between Volume Management's symbolic name for a device, called a <i>symname</i> , and the <i>/dev pathname</i> for that same device.</p> <p>volmgt_symname () converts a supplied <i>/dev pathname</i> to a <i>symname</i> , Volume Management's idea of that device's symbolic name (see volfs(7FS) for a description of Volume Management symbolic names).</p> <p>volmgt_symdev () does the opposite conversion, converting between a <i>symname</i> , Volume Management's idea of a device's symbolic name for a volume, to the <i>/dev pathname</i> for that device.</p> |
| RETURN VALUES | <p>volmgt_symname () returns the symbolic name for the device pathname supplied, and volmgt_symdev () returns the device pathname for the supplied symbolic name.</p> <p>These strings are allocated upon success, and therefore must be freed by the caller when they are no longer needed (see free(3C)).</p> |
| ERRORS | <p>volmgt_symname () can fail, returning a null string pointer, if a stat(2) of the supplied <i>pathname</i> fails, or if an open(2) of <i>/dev/volctl</i> fails, or if any of the following is true:</p> <p>ENXIO Volume Management is not running.</p> <p>EINTR An interrupt signal was detected while trying to convert the supplied <i>pathname</i> to a <i>symname</i> .</p> <p>volmgt_symdev () can fail if an open(2) of <i>/dev/volctl</i> fails, or if any of the following is true:</p> |

ENXIO Volume Management is not running.

EINTR An interrupt signal was detected while trying to convert the supplied *symname* to a */dev pathname*.

EXAMPLES**EXAMPLE 1** Testing floppies.

The following tests how many floppies Volume Management currently sees in floppy drives (up to 10):

```
for (i=0; i < 10; i++) {
    (void) sprintf(path, "floppy%d", i);
    if (volmgt_symdev(path) != NULL) {
        (void) printf("volume %s is in drive %d\
",
                    path, i);
    }
}
```

EXAMPLE 2 Finding the symbolic name.

This code finds out what symbolic name (if any) Volume Management has for */dev/rdisk/c0t6d0s2*:

```
if ((nm = volmgt_symname("/dev/rdisk/c0t6d0s2")) == NULL) {
    (void) printf("path not managed\
");
} else {
    (void) printf("path managed as %s\
", nm);
}
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

cc(1B), **vold(1M)**, **open(2)**, **stat(2)**, **free(3C)**, **malloc(3C)**, **volmgt_check(3X)**, **volmgt_inuse(3X)**, **volmgt_running(3X)**, **attributes(5)**, **volfs(7FS)**

NOTES

These routines only work when Volume Management is running.

BUGS

There should be a straightforward way to query Volume Management for a list of all media types it's managing, and how many of each type are being managed.

| | |
|----------------------|---|
| NAME | vpfmt – display error message in standard format and pass to logging and monitoring services |
| SYNOPSIS | <pre>#include <pfmt.h> #include <stdarg.h> int vpfmt(FILE *stream, long flag, const char *format, va_list ap);</pre> |
| DESCRIPTION | <p>The vpfmt() function is identical to pfmt(3C), except that it is called with an argument list as defined by <code><stdarg.h></code>.</p> <p>The <code><stdarg.h></code> header defines the type <code>va_list</code> and a set of macros for advancing through a list of arguments whose number and types may vary. The <code>ap</code> argument is of type <code>va_list</code>. This argument is used with the <code><stdarg.h></code> macros va_start(), va_arg(), and va_end(). See stdarg(5). The example in the EXAMPLES section below demonstrates their use with vpfmt().</p> |
| RETURN VALUES | Upon successful completion, vpfmt() returns the number of bytes transmitted. Otherwise, <code>-1</code> is returned if there was a write error to <code>stream</code> . |
| EXAMPLES | <p>EXAMPLE 1 Use of vpfmt() to write an error routine.</p> <p>The following example demonstrates how vpfmt() could be used to write an error() routine. The va_alist() macro is used as the parameter list in a function definition. The <code>va_start(ap, ...)</code> call, where <code>ap</code> is of type <code>va_list</code>, must be invoked before any attempt to traverse and access unnamed arguments. Calls to <code>va_arg(ap, atype)</code> traverse the argument list. Each execution of va_arg() expands to an expression with the value and type of the next argument in the list <code>ap</code>, which is the same object initialized by va_start(). The <code>atype</code> argument is the type that the returned argument is expected to be. The <code>va_end(ap)</code> macro must be invoked when all desired arguments have been accessed. The argument list in <code>ap</code> can be traversed again if va_start() is called again after va_end(). In the example below, va_arg() is executed first to retrieve the format string passed to error(). The remaining error() arguments (<code>arg1</code>, <code>arg2</code>, ...) are passed to vpfmt() in the argument <code>ap</code>.</p> <pre>#include <pfmt.h> #include <stdarg.h> /* * error should be called like * error(format, arg1, ...); */ void error(...) { va_list ap; char *format; va_start(ap,); format = va_arg(ap, char *); (void) vpfmt(stderr, MM_ERROR, format, ap); va_end(ap); }</pre> |

```
        (void) abort();  
    }
```

USAGE Since **vpfmt()** uses **gettxt(3C)**, it is recommended that **vpfmt()** not be used.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **gettxt(3C)**, **pfmt(3C)**, **attributes(5)**, **stdarg(5)**

| | |
|----------------------|---|
| NAME | vprintf, vfprintf, vsprintf, vsnprintf – print formatted output of a variable argument list |
| SYNOPSIS | <pre>#include <stdio.h> #include <stdarg.h> int vprintf(const char * format, va_list ap); int vfprintf(FILE * stream, const char * format, va_list ap); int vsprintf(char * s, const char * format, va_list ap); int vsnprintf(char * s, size_t n, const char * format, va_list ap);</pre> |
| DESCRIPTION | <p>The vprintf(), vfprintf(), vsprintf() and vsnprintf() functions are the same as printf(), fprintf(), sprintf(), and snprintf(), respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined in the <code><stdarg.h></code> header. See printf(3S) and stdarg(5).</p> <p>The <code><stdarg.h></code> header defines the type <code>va_list</code> and a set of macros for advancing through a list of arguments whose number and types may vary. The argument <i>ap</i> to the <code>vprint</code> family of functions is of type <code>va_list</code>. This argument is used with the <code><stdarg.h></code> header file macros va_start(), va_arg(), and va_end() (see stdarg(5)). The EXAMPLES section below demonstrates the use of va_start() and va_end() with vprintf().</p> <p>The macro va_alist() is used as the parameter list in a function definition, as in the function called error() in the example below. The macro <code>va_start(ap, parmN)</code>, where <i>ap</i> is of type <code>va_list</code> and <i>parmN</i> is the rightmost parameter (just before <code>...</code>), must be called before any attempt to traverse and access unnamed arguments is made. The <code>va_end(ap)</code> macro must be invoked when all desired arguments have been accessed. The argument list in <i>ap</i> can be traversed again if va_start() is called again after va_end(). In the example below, the error() arguments (<i>arg1</i>, <i>arg2</i>, ...) are passed to vfprintf() in the argument <i>ap</i>.</p> |
| RETURN VALUES | The vprintf() , vfprintf() , and vsprintf() functions return the number of characters transmitted (not including <code>\\0</code> in the case of vsprintf()). The vsnprintf() function returns the number of characters formatted, that is, the number of characters that would have been written to the buffer if it were large enough. Each function returns a negative value if an output error was encountered. |
| ERRORS | <p>The vprintf() and vfprintf() functions will fail if either the <i>stream</i> is unbuffered or the <i>stream</i>'s buffer needed to be flushed and:</p> <p>EFBIG The file is a regular file and an attempt was made to write at or beyond the offset maximum.</p> |

EXAMPLES**EXAMPLE 1** Using **vprintf()** to write an error routine.

The following demonstrates how **vprintf()** could be used to write an error routine:

```
#include <stdio.h>
#include <stdarg.h>
. . .
/*
 *   error should be called like
 *       error(function_name, format, arg1, ...);
 */
void error(char *function_name, char *format, ...)
{
    va_list ap;
    va_start(ap, format);
    /* print out name of function causing error */
    (void) fprintf(stderr, "ERR in %s: ", function_name);
    /* print out remainder of message */
    (void) vfprintf(stderr, format, ap);
    va_end(ap);
    (void) abort;
}
```

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | See NOTES below. |

SEE ALSO

printf(3S), **attributes(5)**, **stdarg(5)**

NOTES

The **vprintf()**, **vfprintf()**, and **vsprintf()** functions are MT-Safe in multithreaded applications.

NAME vsyslog – log message with a varargs argument list

SYNOPSIS

```
#include <syslog.h>
#include <varargs.h>
```

```
int vsyslog(int priority, const char *message, va_list ap);
```

DESCRIPTION **vsyslog()** is the same as **syslog(3)** except that instead of being called with a variable number of arguments, it is called with an argument list as defined by **varargs(5)**.

EXAMPLES

EXAMPLE 1 Using **vsyslog()** To Write An Error Routine

The following demonstrates how **vsyslog()** can be used to write an error routine.

```
#include <syslog.h>
#include <varargs.h>
. . .
/*
 * error should be called like:
 * error(pri, function_name, format, arg1, arg2...);
 * Note that pri, function_name, and format cannot be declared
 * separately because of the definition of varargs.
 */
/*VARARGS0*/
void
error(va_alist)
va_dcl; {
va_list args;
int pri;
char *message;
    va_start(args);
pri = va_arg(args, int);
/* log name of function causing error */
(void) syslog(pri, "ERROR in %s", va_arg(args, char *));
message = va_arg(args, char *);
/* log remainder of message */
(void) vsyslog(pri, msg, args);
va_end(args);
(void) abort( );
}
```

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO | `syslog(3)`, `attributes(5)`, `varargs(5)`

| | | | | | |
|--------------------|--|---------|---|-----------|---|
| NAME | wait3, wait4 – wait for process to terminate or stop | | | | |
| SYNOPSIS | <pre>#include <sys/wait.h> #include <sys/time.h> #include <sys/resource.h> pid_t wait3(int * statusp, int options, struct rusage * rusage); pid_t wait4(pid_t pid, int * statusp, int options, struct rusage * rusage);</pre> | | | | |
| DESCRIPTION | <p>The wait3() function delays its caller until a signal is received or one of its child processes terminates or stops due to tracing. If any child process has died or stopped due to tracing and this has not already been reported, return is immediate, returning the process ID and status of one of those children. If that child process has died, it is discarded. If there are no children, -1 is returned immediately. If there are only running or stopped but reported children, the calling process is blocked.</p> <p>If <i>statusp</i> is not a null pointer, then on return from a successful wait3() call, the status of the child process is stored in the integer pointed to by <i>statusp</i>. <i>*statusp</i> indicates the cause of termination and other information about the terminated process in the following manner:</p> <ul style="list-style-type: none"> ■ If the low-order 8 bits of <i>*statusp</i> are equal to 0177, the child process has stopped; the 8 bits higher up from the low-order 8 bits of <i>*statusp</i> contain the number of the signal that caused the process to stop. See signal(5). ■ If the low-order 8 bits of <i>*statusp</i> are non-zero and are not equal to 0177, the child process terminated due to a signal; the low-order 7 bits of <i>*statusp</i> contain the number of the signal that terminated the process. In addition, if the low-order seventh bit of <i>*statusp</i> (that is, bit 0200) is set, a “core image” of the process was produced; see signal(5). ■ Otherwise, the child process terminated due to an exit() call; the 8 bits higher up from the low-order 8 bits of <i>*statusp</i> contain the low-order 8 bits of the argument that the child process passed to exit(); see exit(2). <p>The <i>options</i> argument is constructed from the bitwise inclusive OR of zero or more of the following flags, defined in <code><sys/wait.h></code>:</p> <table border="0" style="width: 100%;"> <tr> <td style="padding-right: 20px;">WNOHANG</td> <td>Execution of the calling process is not suspended if status is not immediately available for any child process.</td> </tr> <tr> <td>WUNTRACED</td> <td>The status of any child processes that are stopped, and whose status has not yet been reported since they stopped, are also reported to the requesting process.</td> </tr> </table> <p>If <i>rusage</i> is not a null pointer, a summary of the resources used by the terminated process and all its children is returned. Only the user time used</p> | WNOHANG | Execution of the calling process is not suspended if status is not immediately available for any child process. | WUNTRACED | The status of any child processes that are stopped, and whose status has not yet been reported since they stopped, are also reported to the requesting process. |
| WNOHANG | Execution of the calling process is not suspended if status is not immediately available for any child process. | | | | |
| WUNTRACED | The status of any child processes that are stopped, and whose status has not yet been reported since they stopped, are also reported to the requesting process. | | | | |

and the system time used are currently available. They are returned in the `ru_utime` and `ru_stime`, members of the `rusage` structure, respectively.

When the `WNOHANG` option is specified and no processes have status to report, `wait3()` returns 0. The `WNOHANG` and `WUNTRACED` options may be combined by the bitwise OR operation of the two values.

The `wait4()` function is an extended interface. With a `pid` argument of 0, it is equivalent to `wait3()`. If `pid` has a nonzero value, then `wait4()` returns status only for the indicated process ID, but not for any other child processes. The status can be evaluated using the macros defined by `wstat(5)`.

RETURN VALUES

If `wait3()` or `wait4()` returns due to a stopped or terminated child process, the process ID of the child is returned to the calling process. Otherwise, `-1` is returned and `errno` is set to indicate the error.

If `wait3()` or `wait4()` return due to the delivery of a signal to the calling process, `-1` is returned and `errno` is set to `EINTR`. If `WNOHANG` was set in `options`, it has at least one child process specified by `pid` for which status is not available, and status is not available for any process specified by `pid`, `0` is returned. Otherwise, `-1` is returned and `errno` is set to indicate the error.

The `wait3()` and `wait4()` functions return `0` if `WNOHANG` is specified and there are no stopped or exited children, and return the process ID of the child process if they return due to a stopped or terminated child process. Otherwise, they return `-1` and set `errno` to indicate the error.

ERRORS

The `wait3()` and `wait4()` functions will fail and return immediately if:

ECHILD The calling process has no existing unwaited-for child processes.

EFAULT The `statusp` or `rusage` arguments point to an illegal address.

EINTR The function was interrupted by a signal. The value of the location pointed to by `statusp` is undefined.

EINVAL The value of `options` is not valid.

The `wait4()` function may fail if:

ECHILD The process specified by `pid` does not exist or is not a child of the calling process.

The `wait3()` and `wait4()` functions will terminate prematurely, return `-1`, and set `errno` to `EINTR` upon the arrival of a signal whose `SA_RESTART` bit in its flags field is not set (see `sigaction(2)`).

SEE ALSO

`kill(1)`, `exit(2)`, `wait(2)`, `waitid(2)`, `waitpid(2)`, `getrusage(3C)`, `signal(3C)`, `proc(4)`, `signal(5)`, `wstat(5)`

NOTES

If a parent process terminates without waiting on its children, the initialization process (process ID = 1) inherits the children.

The **wait3()** and **wait4()** functions are automatically restarted when a process receives a signal while awaiting termination of a child process, unless the `SA_RESTART` bit is not set in the flags for that signal.

NAME | wait, wait3, wait4, waitpid, WIFSTOPPED, WIFSIGNALED, WIFEXITED –
wait for process to terminate or stop

SYNOPSIS | /usr/ucb/cc
[
 flag
 ...]
 file
 ...
 #include <sys/wait.h>

 int wait(*statusp*);

 int *
 statusp
 ;

 int waitpid(*pid, statusp, options*);

 int
 pid
 ;
 int *
 statusp
 ;
 int
 options
 ;

 #include <sys/time.h>
 #include <sys/resource.h>

 int wait3(*statusp, options, rusage*);

 int *
 statusp
 ;
 int
 options
 ;
 struct rusage *
 rusage
 ;

 int wait4(*pid, statusp, options, rusage*);

 int
 pid

```

;
int *
statusp
;
int
options
;
struct rusage *
rusage
;

WIFSTOPPED( status);

int
status
;

WIFSIGNALED( status);

int
status
;

WIFEXITED( status);

int
status
;

```

DESCRIPTION

wait() delays its caller until a signal is received or one of its child processes terminates or stops due to tracing. If any child process has died or stopped due to tracing and this has not been reported using **wait()**, return is immediate, returning the process ID and exit status of one of those children. If that child process has died, it is discarded. If there are no children, return is immediate with the value `-1` returned. If there are only running or stopped but reported children, the calling process is blocked.

If *status* is not a `NULL` pointer, then on return from a successful **wait()** call the status of the child process whose process ID is the return value of **wait()** is stored in the **wait()** union pointed to by *status*. The `w_status` member of that union is an `int`; it indicates the cause of termination and other information about the terminated process in the following manner:

- If the low-order 8 bits of `w_status` are equal to 0177, the child process has stopped; the 8 bits higher up from the low-order 8 bits of `w_status` contain the number of the signal that caused the process to stop. See **ptrace(2)** and **sigvec(3B)**.

- If the low-order 8 bits of `w_status` are non-zero and are not equal to 0177, the child process terminated due to a signal; the low-order 7 bits of `w_status` contain the number of the signal that terminated the process. In addition, if the low-order seventh bit of `w_status` (that is, bit 0200) is set, a “core image” of the process was produced; see `sigvec(3B)`.
- Otherwise, the child process terminated due to an `exit()` call; the 8 bits higher up from the low-order 8 bits of `w_status` contain the low-order 8 bits of the argument that the child process passed to `exit()`; see `exit(2)`.

`waitpid()` behaves identically to `wait()` if `pid` has a value of `-1` and `options` has a value of zero. Otherwise, the behavior of `waitpid()` is modified by the values of `pid` and `options` as follows:

`pid` specifies a set of child processes for which status is requested. `waitpid()` only returns the status of a child process from this set.

- If `pid` is equal to `-1`, status is requested for any child process. In this respect, `waitpid()` is then equivalent to `wait()`.
- If `pid` is greater than zero, it specifies the process ID of a single child process for which status is requested.
- If `pid` is equal to zero, status is requested for any child process whose process group ID is equal to that of the calling process.
- If `pid` is less than `-1`, status is requested for any child process whose process group ID is equal to the absolute value of `pid`.

`options` is constructed from the bitwise inclusive OR of zero or more of the following flags, defined in the header `<sys/wait.h>`:

`WNOHANG` `waitpid()` does not suspend execution of the calling process if status is not immediately available for one of the child processes specified by `pid`.

`WUNTRACED` The status of any child processes specified by `pid` that are stopped, and whose status has not yet been reported since they stopped, are also reported to the requesting process.

`wait3()` is an alternate interface that allows both non-blocking status collection and the collection of the status of children stopped by any means. The `status` parameter is defined as above. The `options` parameter is used to indicate the call should not block if there are no processes that have status to report (`WNOHANG`), and/or that children of the current process that are stopped due to a `SIGTTIN`, `SIGTTOU`, `SIGTSTP`, or `SIGSTOP` signal are eligible to have their status reported as well (`WUNTRACED`). A terminated child is discarded after it reports status, and a stopped process will not report its status more than once. If `rusage` is not a `NULL` pointer, a summary of the resources used by the terminated process and all its children is returned. Only the user time used

and the system time used are currently available. They are returned in `rusage.ru_utime` and `rusage.ru_stime`, respectively.

When the `WNOHANG` option is specified and no processes have status to report, **wait3()** returns 0. The `WNOHANG` and `WUNTRACED` options may be combined by ORing the two values.

wait4() is another alternate interface. With a *pid* argument of 0, it is equivalent to **wait3()**. If *pid* has a nonzero value, then **wait4()** returns status only for the indicated process ID, but not for any other child processes.

`WIFSTOPPED`, `WIFSIGNALED`, `WIFEXITED`, are macros that take an argument *status*, of type `int`, as returned by **wait()**, or **wait3()**, or **wait4()**. `WIFSTOPPED` evaluates to true (1) when the process for which the **wait()** call was made is stopped, or to false (0) otherwise. `WIFSIGNALED` evaluates to true when the process was terminated with a signal. `WIFEXITED` evaluates to true when the process exited by using an `exit(2)` call.

RETURN VALUES

If **wait()** or **waitpid()** returns due to a stopped or terminated child process, the process ID of the child is returned to the calling process. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

If **wait()** or **waitpid()** return due to the delivery of a signal to the calling process, a value of -1 is returned and `errno` is set to `EINTR`. If **waitpid()** function was invoked with `WNOHANG` set in *options*, it has at least one child process specified by *pid* for which status is not available, and status is not available for any process specified by *pid*, a value of zero is returned. Otherwise, a value of -1 is returned, and `errno` is set to indicate the error.

`wait3()` and `wait4()` returns 0 if `WNOHANG` is specified and there are no stopped or exited children, and returns the process ID of the child process if it returns due to a stopped or terminated child process. Otherwise, they returns a value of -1 and sets `errno` to indicate the error.

ERRORS

wait(), **wait3()** or **wait4()** will fail and return immediately if one or more of the following are true:

- | | |
|---|--|
| ECHILD | The calling process has no existing unwaited-for child processes. |
| EFAULT | The <i>status</i> or <i>rusage</i> arguments point to an illegal address. |
| waitpid() may set <code>errno</code> to: | |
| ECHILD | The process or process group specified by <i>pid</i> does not exist or is not a child of the calling process. |
| EINTR | The function was interrupted by a signal. The value of the location pointed to by <i>statusp</i> is undefined. |

EINVAL The value of *options* is not valid. **wait()**, and **wait3()**, and **wait4()** will terminate prematurely, return `-1`, and set `errno` to `EINTR` upon the arrival of a signal whose `SV_INTERRUPT` bit in its flags field is set (see `sigvec(3B)` and `siginterrupt(3B)`). `signal(3B)`, sets this bit for any signal it catches.

SEE ALSO

`exit(2)`, `ptrace(2)`, `wait(2)`, `waitpid(2)`, `getrusage(3C)`, `siginterrupt(3B)`, `signal(3B)`, `sigvec(3B)`, `signal(3C)`

NOTES

Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

If a parent process terminates without waiting on its children, the initialization process (process ID = 1) inherits the children.

wait(), and **wait3()**, and **wait4()** are automatically restarted when a process receives a signal while awaiting termination of a child process, unless the `SV_INTERRUPT` bit is set in the flags for that signal.

Calls to **wait()** with an argument of 0 should be cast to type `'int *'`, as in:

```
wait((int *)0)
```

Previous SunOS releases used union `wait *statusp` and union `wait status` in place of `int *statusp` and `int status`. The union contained a member `w_status` that could be treated in the same way as `status`.

Other members of the `wait` union could be used to extract this information more conveniently:

- If the `w_stopval` member had the value `WSTOPPED`, the child process had stopped; the value of the `w_stopsig` member was the signal that stopped the process.
- If the `w_termsig` member was non-zero, the child process terminated due to a signal; the value of the `w_termsig` member was the number of the signal that terminated the process. If the `w_coredump` member was non-zero, a core dump was produced.
- Otherwise, the child process terminated due to a call to `exit()`. The value of the `w_retcode` member was the low-order 8 bits of the argument that the child process passed to `exit()`.

`union wait` is obsolete in light of the new specifications provided by *IEEE Std 1003.1-1988* and endorsed by *SVID89* and *XPG3*. SunOS Release 4.1 supports `union wait` for backward compatibility, but it will disappear in a future release.

| | |
|--------------------|--|
| NAME | watchmalloc, cfree, memalign, valloc – debugging memory allocator |
| SYNOPSIS | <pre>#include <stdlib.h> void * malloc(size_t size); void free(void * ptr); void * realloc(void * ptr, size_t size); void * memalign(size_t alignment, size_t size); void * valloc(size_t size); void * calloc(size_t nelem, size_t elsize); void cfree(void * ptr, size_t nelem, size_t elsize); #include <malloc.h> int mallopt(int cmd, int value); struct mallinfo mallinfo(void);</pre> |
| DESCRIPTION | <p>The collection of malloc() routines in this shared object are an optional replacement for the standard versions of the same routines in the system C library. See malloc(3C) . They provide a more strict interface than the standard versions and enable enforcement of the interface via the watchpoint facility of /proc . See proc(4) .</p> <p>Any dynamically linked program can be run with these routines in place of the standard routines if the following string is present in the environment (see ld.so.1(1)):</p> <pre>LD_PRELOAD=watchmalloc.so.1</pre> <p>The individual routine interfaces are identical to the standard ones as described in malloc(3C) . However, laxities provided in the standard versions are not permitted:</p> <ul style="list-style-type: none"> Memory may not be freed more than once. A pointer to freed memory may not be used in a call to realloc() . A malloc() immediately following a free() will not return the same space. Any reference to memory that has been freed yields undefined results. <p>To enforce these restrictions partially, without great loss in speed as compared to the watchpoint facility described below, a freed block of memory is overwritten with the pattern <code>0xdeadbeef</code> before returning from free() .</p> |

WATCHPOINTS

malloc() returns with the allocated memory filled with the pattern `0xbaddcafe` as a precaution against programs incorrectly expecting to receive back unmodified memory from the last **free()**. (`calloc()` always returns with the memory zero-filled.)

Entry points for **mallopt()** and **mallinfo()** are provided as empty routines, and are present only because some **malloc()** implementations provide them.

The watchpoint facility of `/proc` can be applied by a process to itself. The routines in `watchmalloc.so.1` use this feature if the following string is present in the environment:

```
MALLOC_DEBUG=WATCH
```

This causes every block of freed memory to be covered with `WA_WRITE` watched areas. If the program attempts to write any part of freed memory, it will trigger a watchpoint trap, which will result in a `SIGTRAP` signal, which normally results in a program core dump.

A header is maintained before each block of allocated memory. Each header is covered with a watched area, thereby providing a red zone before and after each block of allocated memory (the header for the subsequent memory block serves as the trailing red zone for its preceding memory block). Writing just before or just after a memory block returned by **malloc()** will trigger a watchpoint trap.

Watchpoints incur a large performance penalty. Requesting `MALLOC_DEBUG=WATCH` can cause the program to run 10 to 100 times slower, depending on the use made of allocated memory.

Further options are enabled by specifying a comma-separated string of options:

| | |
|-------|--|
| WATCH | Enables <code>WA_WRITE</code> watched areas as described above. |
| RW | Enables both <code>WA_READ</code> and <code>WA_WRITE</code> watched areas. An attempt either to read or write freed memory or the red zones will trigger a watchpoint trap. This incurs even more overhead and can cause the program to run up to 1000 times slower. |
| STOP | The process will stop showing a <code>FLTWATCH</code> machine fault if it triggers a watchpoint trap, rather than dumping core with a <code>SIGTRAP</code> signal. This allows a debugger to be attached to the live process at the point where it underwent the |

watchpoint trap. Also, the various `/proc` tools described in `proc(1)` can be used to examine the stopped process. One of `WATCH` or `RW` must be specified, else the watchpoint facility is not engaged. `RW` overrides `WATCH`. Unrecognized options are silently ignored.

LIMITATIONS

Interposition of `watchmalloc.so.1` fails innocuously if the target program is statically linked with respect to its `malloc()` routines. The system-supplied libraries `-lmalloc` and `-lbsdmalloc` are provided only in archive format and therefore programs linked with these libraries are immune to the interposition of `watchmalloc.so.1`.

FILES

`/usr/lib/watchmalloc.so.1`

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`proc(1)`, `bsdmalloc(3X)`, `calloc(3C)`, `free(3C)`, `malloc(3C)`, `malloc(3X)`, `mapmalloc(3X)`, `memalign(3C)`, `realloc(3C)`, `valloc(3C)`, `libmapmalloc(4)`, `proc(4)`, `attributes(5)`

| | |
|----------------------|--|
| NAME | wrtomb – convert a wide-character code to a character (restartable) |
| SYNOPSIS | <pre>#include <stdio.h> size_t wrtomb(char *s, wchar_t wc, mbstate_t *ps);</pre> |
| DESCRIPTION | <p>If <i>s</i> is a null pointer, the wrtomb() function is equivalent to the call:</p> <pre>wrtomb(buf, L'\0', ps)</pre> <p>where <i>buf</i> is an internal buffer.</p> <p>If <i>s</i> is not a null pointer, the wrtomb() function determines the number of bytes needed to represent the character that corresponds to the wide-character given by <i>wc</i> (including any shift sequences), and stores the resulting bytes in the array whose first element is pointed to by <i>s</i>. At most MB_CUR_MAX bytes are stored. If <i>wc</i> is a null wide-character, a null byte is stored, preceded by any shift sequence needed to restore the initial shift state. The resulting state described is the initial conversion state.</p> <p>If <i>ps</i> is a null pointer, the wrtomb() function uses its own internal <code>mbstate_t</code> object, which is initialized at program startup to the initial conversion state. Otherwise, the <code>mbstate_t</code> object pointed to by <i>ps</i> is used to completely describe the current conversion state of the associated character sequence. Solaris will behave as if no function defined in the Solaris Reference Manual calls wrtomb().</p> <p>The behavior of this function is affected by the LC_CTYPE category of the current locale. See environ(5).</p> |
| RETURN VALUES | The wrtomb() function returns the number of bytes stored in the array object (including any shift sequences). When <i>wc</i> is not a valid wide-character, an encoding error occurs. In this case, the function stores the value of the macros EILSEQ in <code>errno</code> and returns <code>(size_t)-1</code> ; the conversion state is undefined. |
| ERRORS | <p>The wrtomb() function may fail if:</p> <p>EINVAL The <i>ps</i> argument points to an object that contains an invalid conversion state.</p> <p>EILSEQ Invalid wide-character code is detected.</p> |
| USAGE | If <i>ps</i> is not a null pointer, wrtomb() uses the <code>mbstate_t</code> object pointed to by <i>ps</i> and the function can be used safely in multithreaded applications, as long as setlocale(3C) is not being called to change the locale. If <i>ps</i> is a null |

pointer, **wrtomb()** uses its internal `mbstate_t` object and the function is Unsafe in multithreaded applications.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | See NOTES below |

SEE ALSO

mbsinit(3C), **setlocale(3C)**, **attributes(5)**, **environ(5)**

| NAME | wscoll, wscoll – wide character string comparison using collating information | | | | | | |
|----------------------|---|----------------|-----------------|----------|-------------------------|-----|---------|
| SYNOPSIS | <pre>#include <wchar.h> int wscoll(const wchar_t * <i>ws1</i>, const wchar_t * <i>ws2</i>); int wscoll(const wchar_t * <i>ws1</i>, const wchar_t * <i>ws2</i>);</pre> | | | | | | |
| DESCRIPTION | The wscoll() and wscoll() functions compare the wide character string pointed to by <i>ws1</i> to the wide character string pointed to by <i>ws2</i> , both interpreted as appropriate to the LC_COLLATE category of the current locale. | | | | | | |
| RETURN VALUES | Upon successful completion, wscoll() and wscoll() return an integer greater than, equal to, or less than 0, depending upon whether the wide character string pointed to by <i>ws1</i> is greater than, equal to, or less than the wide character string pointed to by <i>ws2</i> , when both are interpreted as appropriate to the current locale. On error, wscoll() and wscoll() may set <code>errno</code> , but no return value is reserved to indicate an error. | | | | | | |
| ERRORS | <p>The wscoll() and wscoll() functions may fail if:</p> <p>EINVAL The <i>ws1</i> or <i>ws2</i> arguments contain wide character codes outside the domain of the collating sequence.</p> <p>ENOSYS The function is not supported.</p> | | | | | | |
| USAGE | <p>Because no return value is reserved to indicate an error, an application wishing to check for error situations should set <code>errno</code> to 0, call either wscoll() or wscoll(), then check <code>errno</code> and if it is non-zero, assume an error has occurred.</p> <p>The wcsxfrm(3C) and wscmp(3C) functions should be used for sorting large lists.</p> <p>The wscoll() and wscoll() functions can be used safely in multithreaded applications as long as setlocale(3C) is not being called to change the locale.</p> | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe with exceptions</td> </tr> <tr> <td>CSI</td> <td>Enabled</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe with exceptions | CSI | Enabled |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| MT-Level | MT-Safe with exceptions | | | | | | |
| CSI | Enabled | | | | | | |
| SEE ALSO | setlocale(3C) , wscmp(3C) , wcsxfrm(3C) , attributes(5) | | | | | | |

| NAME | wcsftime – convert date and time to wide character string | | | | | | |
|------------------------------------|--|----------------|-----------------|----------|-------------------------|-----|---------|
| SYNOPSIS | #include <wchar.h> | | | | | | |
| XPG4 and SUS | size_t wcsftime (wchar_t *wcs, size_t <i>maxsize</i> , const char * <i>format</i> , const struct tm * <i>timptr</i>); | | | | | | |
| Default and other standards | size_t wcsftime (wchar_t *wcs, size_t <i>maxsize</i> , const wchar_t * <i>format</i> , const struct tm * <i>timptr</i>); | | | | | | |
| DESCRIPTION | <p>The wcsftime() function is equivalent to the strftime(3C) function, except that:</p> <ul style="list-style-type: none"> ■ The argument <i>wcs</i> points to the initial element of an array of wide-characters into which the generated output is to be placed. ■ The argument <i>maxsize</i> indicates the maximum number of wide-characters to be placed in the output array. ■ The argument <i>format</i> is a wide-character string and the conversion specifications are replaced by corresponding sequences of wide-characters. ■ The return value indicates the number of wide-characters placed in the output array. <p>If copying takes place between objects that overlap, the behavior is undefined.</p> | | | | | | |
| RETURN VALUES | <p>If the total number of resulting wide character codes (including the terminating null wide-character code) is no more than <i>maxsize</i>, wcsftime() returns the number of wide-character codes placed into the array pointed to by <i>wcs</i>, not including the terminating null wide-character code. Otherwise, 0 is returned and the contents of the array are indeterminate.</p> <p>The wcftime() function uses malloc(3C) and should malloc() fail, errno will be set by malloc().</p> | | | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe with exceptions</td> </tr> <tr> <td>CSI</td> <td>Enabled</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe with exceptions | CSI | Enabled |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| MT-Level | MT-Safe with exceptions | | | | | | |
| CSI | Enabled | | | | | | |
| SEE ALSO | malloc(3C) , setlocale(3C) , strftime(3C) , attributes(5) , standards(5) | | | | | | |
| NOTES | The wcsftime() function can be used safely in multithreaded applications, as long as setlocale(3C) is not being called to change the locale. | | | | | | |

| | | | | | |
|----------------------|---|---------------|---|---------------|---|
| NAME | wcsrtombs – convert a wide-character string to a character string (restartable) | | | | |
| SYNOPSIS | <pre>#include <wchar.h> size_t wcsrtombs(char *dst, const wchar_t **src, size_t len, mbstate_t *ps);</pre> | | | | |
| DESCRIPTION | <p>The wcsrtombs() function converts a sequence of wide-characters from the array indirectly pointed to by <i>src</i> into a sequence of corresponding characters, beginning in the conversion state described by the object pointed to by <i>ps</i>. If <i>dst</i> is not a null pointer, the converted characters are then stored into the array pointed to by <i>dst</i>. Conversion continues up to and including a terminating null wide-character, which is also stored. Conversion stops earlier in the following cases:</p> <ul style="list-style-type: none"> ■ When a code is reached that does not correspond to a valid character. ■ When the next character would exceed the limit of <i>len</i> total bytes to be stored in the array pointed to by <i>dst</i> (and <i>dst</i> is not a null pointer). <p>Each conversion takes place as if by a call to the wcrtomb() function.</p> <p>If <i>dst</i> is not a null pointer, the pointer object pointed to by <i>src</i> is assigned either a null pointer (if conversion stopped due to reaching a terminating null wide-character) or the address just past the last wide-character converted (if any). If conversion stopped due to reaching a terminating null wide-character, the resulting state described is the initial conversion state.</p> <p>If <i>ps</i> is a null pointer, the wcsrtombs() function uses its own internal <code>mbstate_t</code> object, which is initialized at program startup to the initial conversion state. Otherwise, the <code>mbstate_t</code> object pointed to by <i>ps</i> is used to completely describe the current conversion state of the associated character sequence. Solaris will behave as if no function defined in the Solaris Reference Manual calls wcsrtombs().</p> <p>The behavior of this function is affected by the LC_CTYPE category of the current locale. See environ(5).</p> | | | | |
| RETURN VALUES | <p>If conversion stops because a code is reached that does not correspond to a valid character, an encoding error occurs. In this case, the wcsrtombs() function stores the value of the macro <code>EILSEQ</code> in <code>errno</code> and returns $(\text{size_t})-1$; the conversion state is undefined. Otherwise, it returns the number of bytes in the resulting character sequence, not including the terminating null (if any).</p> | | | | |
| ERRORS | <p>The wcsrtombs() function may fail if:</p> <table border="0" style="width: 100%;"> <tr> <td style="padding-right: 20px;">EINVAL</td> <td>The <i>ps</i> argument points to an object that contains an invalid conversion state.</td> </tr> <tr> <td>EILSEQ</td> <td>A wide-character code does not correspond to a valid character.</td> </tr> </table> | EINVAL | The <i>ps</i> argument points to an object that contains an invalid conversion state. | EILSEQ | A wide-character code does not correspond to a valid character. |
| EINVAL | The <i>ps</i> argument points to an object that contains an invalid conversion state. | | | | |
| EILSEQ | A wide-character code does not correspond to a valid character. | | | | |

USAGE If *ps* is not a null pointer, **wcsrtombs()** uses the `mbstate_t` object pointed to by *ps* and the function can be used safely in multithreaded applications, as long as **setlocale(3C)** is not being called to change the locale. If *ps* is a null pointer, **wcsrtombs()** uses its internal `mbstate_t` object and the function is Unsafe in multithreaded applications.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | See NOTES below |

SEE ALSO **mbsinit(3C)**, **setlocale(3C)**, **wcrtomb(3C)**, **attributes(5)**, **environ(5)**

| NAME | wcsstr – find a wide-character substring | | | | |
|----------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | #include <wchar.h> wchar_t * wcsstr (const wchar_t *ws1, const wchar_t *ws2); | | | | |
| DESCRIPTION | The wcsstr() function locates the first occurrence in the wide-character string pointed to by <i>ws1</i> of the sequence of wide-characters (excluding the terminating null wide-character) in the wide-character string pointed to by <i>ws2</i> . | | | | |
| RETURN VALUES | On successful completion, wcsstr() returns a pointer to the located wide-character string, or a null pointer if the wide-character string is not found. If <i>ws2</i> points to a wide-character string with zero length, the function returns <i>ws1</i> . | | | | |
| ERRORS | No errors are defined. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | wchr(3C) , attributes(5) | | | | |

| | |
|--------------------|--|
| NAME | wcstod, wstod, watof – convert wide character string to double-precision number |
| SYNOPSIS | <pre>#include <wchar.h> double wcstod(const wchar_t * nptr, wchar_t ** endptr); double wstod(const wchar_t * nptr, w char_t ** endptr); double watof(wchar_t * nptr);</pre> |
| DESCRIPTION | <p>The wcstod() and wstod() functions convert the initial portion of the wide character string pointed to by <i>nptr</i> to <code>double</code> representation. They first decompose the input wide character string into three parts: an initial, possibly empty, sequence of white-space wide character codes (as specified by isspace(3C)); a subject sequence interpreted as a floating-point constant; and a final wide-character string of one or more unrecognised wide-character codes, including the terminating null wide character code of the input wide character string. They then attempt to convert the subject sequence to a floating-point number, and return the result.</p> <p>The expected form of the subject sequence is an optional '+' or '-' sign, then a non-empty sequence of digits optionally containing a radix, then an optional exponent part. An exponent part consists of 'e' or 'E', followed by an optional sign, followed by one or more decimal digits. The subject sequence is defined as the longest initial subsequence of the input wide character string, starting with the first non-white-space wide-character code, that is of the expected form. The subject sequence contains no wide-character codes if the input wide character string is empty or consists entirely of white-space wide-character codes, or if the first wide-character code that is not white space other than a sign, a digit or a radix.</p> <p>If the subject sequence has the expected form, the sequence of wide-character codes starting with the first digit or the radix (whichever occurs first) is interpreted as a floating constant as defined in the C language, except that the radix is used in place of a period, and that if neither an exponent part nor a radix appears, a radix is assumed to follow the last digit in the wide character string. If the subject sequence begins with a minus sign (-), the value resulting from the conversion is negated. A pointer to the final wide character string is stored in the object pointed to by <i>endptr</i>, provided that <i>endptr</i> is not a null pointer.</p> <p>The radix is defined in the program's locale (category <code>LC_NUMERIC</code>). In the POSIX locale, or in a locale where the radix is not defined, the radix defaults to a period (.).</p> <p>In other than the POSIX locale, other implementation-dependent subject sequence forms may be accepted.</p> |

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of *nptr* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

The `watof(str)` function is equivalent to `wstod(str, (wchar_t **)NULL)`.

RETURN VALUES

The `wcstod()` and `wstod()` functions return the converted value, if any. If no conversion could be performed, 0 is returned and `errno` may be set to `EINVAL`.

If the correct value is outside the range of representable values, `±HUGE_VAL` is returned (according to the sign of the value), and `errno` is set to `ERANGE`.

If the correct value would cause underflow, 0 is returned, and `errno` is set to `ERANGE`.

ERRORS

The `wcstod()` and `wstod()` functions will fail if:

ERANGE The value to be returned would cause overflow or underflow.

The `wcstod()` and `wstod()` functions may fail if:

EINVAL No conversion could be performed.

USAGE

Because 0 is returned on error and is also a valid return on success, an application wishing to check for error situations should set `errno` to 0 call `wcstod()` or `wstod()`, then check `errno` and if it is non-zero, assume an error has occurred.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`iswspace(3C)`, `localeconv(3C)`, `scanf(3S)`, `setlocale(3C)`, `wcstol(3C)`, `attributes(5)`

| | |
|--------------------|--|
| NAME | wcstol, wstol, watol, watoll, watoi – convert wide character string to long integer |
| SYNOPSIS | <pre>#include <wchar.h> long int wcstol(const wchar_t * nptr, wchar_t ** endptr, int base); #include <wdec.h> long int wstol(const wchar_t * nptr, wchar_t ** endptr, int base); long watol(wchar_t * nptr); long long watoll(wchar_t * nptr); int watoi(wchar_t * nptr);</pre> |
| DESCRIPTION | <p>The wcstol() and wstol() functions convert the initial portion of the wide character string pointed to by <i>nptr</i> to <code>long int</code> representation. They first decompose the input wide character string into three parts: an initial, possibly empty, sequence of white-space wide-character codes (as specified by iswspace(3C)), a subject sequence interpreted as an integer represented in some radix determined by the value of <i>base</i>; and a final wide character string of one or more unrecognised wide character codes, including the terminating null wide-character code of the input wide character string. They then attempt to convert the subject sequence to an integer, and return the result.</p> <p>If the value of <i>base</i> is 0, the expected form of the subject sequence is that of a decimal constant, octal constant or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7' only. A hexadecimal constant consists of the prefix '0x' or '0X' followed by a sequence of the decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.</p> <p>If the value of <i>base</i> is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by <i>base</i>, optionally preceded by a '+' or '-' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of <i>base</i> are permitted. If the value of <i>base</i> is 16, the wide-character code representations of '0x' or '0X' may optionally precede the sequence of letters and digits, following the sign if present.</p> <p>The subject sequence is defined as the longest initial subsequence of the input wide character string, starting with the first non-white-space wide-character code, that is of the expected form. The subject sequence contains no wide-character codes if the input wide character string is empty or consists</p> |

entirely of white-space wide-character code, or if the first non-white-space wide-character code is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of *base* is 0, the sequence of wide-character codes starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it is used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus sign (-), the value resulting from the conversion is negated. A pointer to the final wide character string is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

In other than the POSIX locale, additional implementation-dependent subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of *nptr* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

The **watol()** function is equivalent to `wstol(str, (wchar_t **)NULL, 10)`.

The **watoll()** function is the long-long (double long) version of **watol()**.

The **watoi()** function is equivalent to `(int)watol()`.

RETURN VALUES

Upon successful completion, **wcstol()** and **wstol()** return the converted value, if any. If no conversion could be performed, 0 is returned, and `errno` may be set to indicate the error. If the correct value is outside the range of representable values, `{LONG_MAX}` or `{LONG_MIN}` is returned (according to the sign of the value), and `errno` is set to `ERANGE`.

ERRORS

The **wcstol()** and **wstol()** functions will fail if:

EINVAL The value of *base* is not supported.

ERANGE The value to be returned is not representable.

The **wcstol()** and **wstol()** functions may fail if:

EINVAL No conversion could be performed.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

iswalphabet(3C), **iswspace(3C)**, **scanf(3S)**, **wcstod(3C)**, **attributes(5)**

NOTES

Because 0 , {LONG_MIN} , and {LONG_MAX} are returned on error and are also valid returns on success, an application wishing to check for error situations should set `errno` to 0 , call **wcstol()** or `wstol()` , then check `errno` and if it is non-zero assume an error has occurred.

Truncation from `long long` to `long` can take place upon assignment or by an explicit cast.

| NAME | wcstombs – convert a wide-character string to a character string | | | | | | |
|----------------------|---|----------------|-----------------|----------|---------|-----|---------|
| SYNOPSIS | <pre>#include <stdlib.h> size_t wcstombs(char *s, const wchar_t *pwcs, size_t n);</pre> | | | | | | |
| DESCRIPTION | <p>The wcstombs() function converts the sequence of wide-character codes from the array pointed to by <i>pwcs</i> into a sequence of characters and stores these characters into the array pointed to by <i>s</i>, stopping if a character would exceed the limit of <i>n</i> total bytes or if a null byte is stored. Each wide-character code is converted as if by a call to wctomb(3C).</p> <p>The behavior of this function is affected by the LC_CTYPE category of the current locale.</p> <p>No more than <i>n</i> bytes will be modified in the array pointed to by <i>s</i>. If copying takes place between objects that overlap, the behavior is undefined. If <i>s</i> is a null pointer, wcstombs() returns the length required to convert the entire array regardless of the value of <i>n</i>, but no values are stored.</p> | | | | | | |
| RETURN VALUES | <p>If a wide-character code is encountered that does not correspond to a valid character (of one or more bytes each), wcstombs() returns <code>(size_t)-1</code>. Otherwise, wcstombs() returns the number of bytes stored in the character array, not including any terminating null byte. The array will not be null-terminated if the value returned is <i>n</i>.</p> | | | | | | |
| ERRORS | <p>The wcstombs() function may fail if:</p> <p>EILSEQ A wide-character code does not correspond to a valid character.</p> | | | | | | |
| ATTRIBUTES | <p>See attributes(5) for descriptions of the following attributes:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> <tr> <td>CSI</td> <td>Enabled</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe | CSI | Enabled |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| MT-Level | MT-Safe | | | | | | |
| CSI | Enabled | | | | | | |
| SEE ALSO | <p>mblen(3C), mbstowcs(3C), mbtowc(3C), setlocale(3C), wctomb(3C), attributes(5)</p> | | | | | | |

| | |
|--------------------|--|
| NAME | wcstoul – convert wide character string to unsigned long |
| SYNOPSIS | <pre>#include <wchar.h> unsigned long int wcstoul(const wchar_t *nptr, wchar_t **endptr, int base);</pre> |
| DESCRIPTION | <p>The wcstoul() function converts the initial portion of the wide character string pointed to by <i>nptr</i> to unsigned long int representation. It first decomposes the input wide-character string into three parts: an initial, possibly empty, sequence of white-space wide-character codes (as specified by the function isspace(3C)); a subject sequence interpreted as an integer represented in some radix determined by the value of <i>base</i>; and a final wide-character string of one or more unrecognized wide character codes, including the terminating null wide-character code of the input wide character string. It then attempts to convert the subject sequence to an unsigned integer, and returns the result.</p> <p>If the value of <i>base</i> is 0, the expected form of the subject sequence is that of a decimal constant, an octal constant, or a hexadecimal constant, any of which may be preceded by a '+' or a '-' sign. A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal constant consists of the prefix '0', optionally followed by a sequence of the digits '0' to '7' only. A hexadecimal constant consists of the prefix '0x' or '0X', followed by a sequence of the decimal digits and letters 'a' (or 'A') to 'f' (or 'F'), with values 10 to 15, respectively.</p> <p>If the value of <i>base</i> is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by <i>base</i>, optionally preceded by a '+' or a '-' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of <i>base</i> are permitted. If the value of <i>base</i> is 16, the wide-character codes '0x' or '0X' may optionally precede the sequence of letters and digits, following the sign, if present.</p> <p>The subject sequence is defined as the longest initial subsequence of the input wide-character string, starting with the first wide-character code that is not a white space and is of the expected form. The subject sequence contains no wide-character codes if the input wide-character string is empty or consists entirely of white-space wide-character codes, or if the first wide-character code that is not a white space is other than a sign or a permissible letter or digit.</p> <p>If the subject sequence has the expected form and the value of <i>base</i> is 0, the sequence of wide-character codes starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of <i>base</i> is between 2 and 36, it is used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A</p> |

pointer to the final wide character string is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

In other than the POSIX locale, additional subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of *nptr* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

Because 0 and `ULONG_MAX` are returned on error and 0 is also a valid return on success, an application wishing to check for error situations should set `errno` to 0, call `wcstoul()`, then check `errno` and if it is non-zero, assume an error has occurred.

RETURN VALUE

Upon successful completion, `wcstoul()` returns the converted value, if any, and does not change the setting of `errno`. If no conversion could be performed, 0 is returned and `errno` may be set to indicate the error. If the correct value is outside the range of representable values, `ULONG_MAX` is returned and `errno` is set to `ERANGE`.

ERRORS

The `wcstoul()` function will fail if:

EINVAL The value of *base* is not supported.

ERANGE The value to be returned is not representable.

The `wcstoul()` function may fail if:

EINVAL No conversion could be performed.

USAGE

Unlike `wcstod(3C)` and `wcstol(3C)`, `wcstoul()` must always return a non-negative number; using the return value of `wcstoul()` for out-of-range numbers with `wcstoul()` could cause more severe problems than just loss of precision if those numbers can ever be negative.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`isspace(3C)`, `iswalpha(3C)`, `scanf(3S)`, `wcstod(3C)`, `wcstol(3C)`, `attributes(5)`

NAME wcstring, wscat, wscat, wcsncat, wsncat, wscmp, wscmp, wcsncmp, wsncmp, wcsncpy, wcsncpy, wcsncpy, wsncpy, wcslen, wslen, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcsprbrk, wspbrk, wcsvcs, wcsspn, wssp, wcspspn, wscspn, wcstok, wstok – wide-character string operations

SYNOPSIS

```
#include <wchar.h>

wchar_t * wscat(wchar_t * ws1, const wchar_t * ws2);

wchar_t * wcsncat(wchar_t * ws1, const wchar_t * ws2, size_t n);

int wscmp(const wchar_t * ws1, const wchar_t * ws2);

int wcsncmp(const wchar_t * ws1, const wchar_t * ws2, size_t n);

wchar_t * wcsncpy(wchar_t * ws1, const wchar_t * ws2);

wchar_t * wcsncpy(wchar_t * ws1, const wchar_t * ws2, size_t n);

size_t wcslen(const wchar_t * ws);

wchar_t * wcschr(const wchar_t * ws, wchat_t wc);

wchar_t * wcsrchr(const wchar_t * ws, wchar_t wc);

wchar_t * wcsprbrk(const wchar_t * ws1, const wchar_t * ws2);

wchar_t * wcsvcs(const wchar_t * ws1, const wchar_t * ws2);

size_t wcsspn(const wchar_t * ws1, const wchar_t * ws2);

size_t wcspspn(const wchar_t * ws1, const wchar_t * ws2);

XPG4 and SUS
wchar_t * wcstok(wchar_t * ws1, const wchar_t * ws2);

Default and other standards
wchar_t * wcstok(wchar_t * ws1, const wchar_t * ws2, wchar_t ** ptr);

#include <widec.h>

wchar_t * wscat(wchar_t * ws1, const wchar_t * ws2);

wchar_t * wsncat(wchar_t * ws1, const wchar_t * ws2, size_t n);

int wscmp(const wchar_t * ws1, const wchar_t * ws2);

int wsncmp(const wchar_t * ws1, const wchar_t * ws2, size_t n);

wchar_t * wcsncpy(wchar_t * ws1, const wchar_t * ws2);

wchar_t * wsncpy(wchar_t * ws1, const wchar_t * ws2, size_t n);

size_t wslen(const wchar_t * ws);

wchar_t * wschr(const wchar_t * ws, wchat_t wc);
```

```

wchar_t * wrchr(const wchar_t * ws, wchar_t wc);
wchar_t * wspbrk(const wchar_t * ws1, const wchar_t * ws2);
size_t wsspncpy(const wchar_t * ws1, const wchar_t * ws2);
size_t wscspncpy(const wchar_t * ws1, const wchar_t * ws2);
wchar_t * wstok(wchar_t * ws1, const wchar_t * ws2);
wchar_t * windex(const wchar_t * ws, wchar_t wc);
wchar_t * wrindex(const wchar_t * ws, wchar_t wc);

```

DESCRIPTION

These functions operate on wide-character strings terminated by `wchar_t` `NULL` characters. During appending or copying, these routines do not check for an overflow condition of the receiving string. In the following, `ws`, `ws1`, and `ws2` point to wide-character strings terminated by a `wchar_t` `NULL`.

wscat(), wscat()

The **wscat()** and **wscat()** functions append a copy of the wide-character string pointed to by `ws2` (including the terminating null wide-character code) to the end of the wide-character string pointed to by `ws1`. The initial wide-character code of `ws2` overwrites the null wide-character code at the end of `ws1`. If copying takes place between objects that overlap, the behavior is undefined. Both functions return `ws1`; no return value is reserved to indicate an error.

wcsncat(), wcsncat()

The **wcsncat()** and **wcsncat()** functions append not more than `n` wide-character codes (a null wide-character code and wide-character codes that follow it are not appended) from the array pointed to by `ws2` to the end of the wide-character string pointed to by `ws1`. The initial wide-character code of `ws2` overwrites the null wide-character code at the end of `ws1`. A terminating null wide-character code is always appended to the result. Both functions return `ws1`; no return value is reserved to indicate an error.

wscmp(), wscmp()

The **wscmp()** and **wscmp()** functions compare the wide-character string pointed to by `ws1` to the wide-character string pointed to by `ws2`. The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of wide-character codes that differ in the objects being compared. Upon completion, both functions return an integer greater than, equal to, or less than zero, if the wide-character string pointed to by `ws1` is greater than, equal to, or less than the wide-character string pointed to by `ws2`.

**wcsncmp(),
wcsncmp()**

The **wcsncmp()** and **wcsncmp()** functions compare not more than `n` wide-character codes (wide-character codes that follow a null wide character code are not compared) from the array pointed to by `ws1` to the array pointed to by `ws2`. The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of wide-character codes that

| | |
|-------------------------------------|---|
| | differ in the objects being compared. Upon successful completion, both functions return an integer greater than, equal to, or less than zero, if the possibly null-terminated array pointed to by <i>ws1</i> is greater than, equal to, or less than the possibly null-terminated array pointed to by <i>ws2</i> . |
| wscpy() , wscpy() | The wscpy() and wscpy() functions copy the wide-character string pointed to by <i>ws2</i> (including the terminating null wide-character code) into the array pointed to by <i>ws1</i> . If copying takes place between objects that overlap, the behavior is undefined. Both functions return <i>ws1</i> ; no return value is reserved to indicate an error. |
| wcsncpy() , wcsncpy() | The wcsncpy() and wcsncpy() functions copy not more than <i>n</i> wide-character codes (wide-character codes that follow a null wide character code are not copied) from the array pointed to by <i>ws2</i> to the array pointed to by <i>ws1</i> . If copying takes place between objects that overlap, the behavior is undefined. If the array pointed to by <i>ws2</i> is a wide-character string that is shorter than <i>n</i> wide-character codes, null wide-character codes are appended to the copy in the array pointed to by <i>ws1</i> , until a total <i>n</i> wide-character codes are written. Both functions return <i>ws1</i> ; no return value is reserved to indicate an error. |
| wcslen() , wcslen() | The wcslen() and wcslen() functions compute the number of wide-character codes in the wide-character string to which <i>ws</i> points, not including the terminating null wide-character code. Both functions return <i>ws</i> ; no return value is reserved to indicate an error. |
| wcschr() , wcschr() | The wcschr() and wcschr() functions locate the first occurrence of <i>wc</i> in the wide-character string pointed to by <i>ws</i> . The value of <i>wc</i> must be a character representable as a type <code>wchar_t</code> and must be a wide-character code corresponding to a valid character in the current locale. The terminating null wide-character code is considered to be part of the wide-character string. Upon completion, both functions return a pointer to the wide-character code, or a null pointer if the wide-character code is not found. |
| wcsrchr() , wcsrchr() | The wcsrchr() and wcsrchr() functions locate the last occurrence of <i>wc</i> in the wide-character string pointed to by <i>ws</i> . The value of <i>wc</i> must be a character representable as a type <code>wchar_t</code> and must be a wide-character code corresponding to a valid character in the current locale. The terminating null wide-character code is considered to be part of the wide-character string. Upon successful completion, both functions return a pointer to the wide-character code, or a null pointer if <i>wc</i> does not occur in the wide-character string. |
| windex() , windex() | The windex() and windex() functions behave the same as wcschr() and wcsrchr() , respectively. |

| | |
|------------------------------------|--|
| wcspbrk() , wspbrk() | The wcspbrk() and wspbrk() functions locate the first occurrence in the wide character string pointed to by <i>ws1</i> of any wide-character code from the wide-character string pointed to by <i>ws2</i> . Upon successful completion, the function returns a pointer to the wide-character code, or a null pointer if no wide-character code from <i>ws2</i> occurs in <i>ws1</i> . |
| wcswcs() | The wcswcs() function locates the first occurrence in the wide-character string pointed to by <i>ws1</i> of the sequence of wide-character codes (excluding the terminating null wide-character code) in the wide-character string pointed to by <i>ws2</i> . Upon successful completion, the function returns a pointer to the located wide-character string, or a null pointer if the wide-character string is not found. If <i>ws2</i> points to a wide-character string with zero length, the function returns <i>ws1</i> . |
| wcsspn() , wsspnl() | The wcsspn() and wsspnl() functions compute the length of the maximum initial segment of the wide-character string pointed to by <i>ws1</i> which consists entirely of wide-character codes from the wide-character string pointed to by <i>ws2</i> . Both functions return the length <i>ws1</i> ; no return value is reserved to indicate an error. |
| wcscspn() , wcspnl() | The wcscspn() and wcspnl() functions compute the length of the maximum initial segment of the wide-character string pointed to by <i>ws1</i> which consists entirely of wide-character codes <i>not</i> from the wide-character string pointed to by <i>ws2</i> . Both functions return the length of the initial substring of <i>ws1</i> ; no return value is reserved to indicate an error. |
| wcstok() , wstok() | A sequence of calls to the wcstok() and wstok() functions break the wide-character string pointed to by <i>ws1</i> into a sequence of tokens, each of which is delimited by a wide-character code from the wide-character string pointed to by <i>ws2</i> . |
| Default and other standards | <p>The third argument points to a caller-provided <code>wchar_t</code> pointer into which the wcstok() function stores information necessary for it to continue scanning the same wide-character string. This argument is not available with the XPG4 and SUS versions of wcstok() , nor is it available with the wstok() function. See standards(5) .</p> <p>The first call in the sequence has <i>ws1</i> as its first argument, and is followed by calls with a null pointer as their first argument. The separator string pointed to by <i>ws2</i> may be different from call to call.</p> <p>The first call in the sequence searches the wide-character string pointed to by <i>ws1</i> for the first wide-character code that is <i>not</i> contained in the current separator string pointed to by <i>ws2</i> . If no such wide-character code is found, then there are no tokens in the wide-character string pointed to by <i>ws1</i> , and</p> |

wcstok() and **wstok()** return a null pointer. If such a wide-character code is found, it is the start of the first token.

The **wcstok()** and **wstok()** functions then search from that point for a wide-character code that is contained in the current separator string. If no such wide-character code is found, the current token extends to the end of the wide-character string pointed to by *ws1*, and subsequent searches for a token will return a null pointer. If such a wide-character code is found, it is overwritten by a null wide character, which terminates the current token. The **wcstok()** and **wstok()** functions save a pointer to the following wide-character code, from which the next search for a token will start.

Each subsequent call, with a null pointer as the value of the first argument, starts searching from the saved pointer and behaves as described above.

Upon successful completion, both functions return a pointer to the first wide-character code of a token. Otherwise, if there is no token, a null pointer is returned.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |
| CSI | Enabled |

SEE ALSO

malloc(3C), **string(3C)**, **wcswidth(3C)**, **wcwidth(3C)**, **attributes(5)**, **standards(5)**

| NAME | wcswidth – number of column positions of a wide-character string | | | | | | |
|----------------------|--|----------------|-----------------|----------|-------------------------|-----|---------|
| SYNOPSIS | <pre>#include <wchar.h> int wcswidth(const wchar_t *pwcs, size_t n);</pre> | | | | | | |
| DESCRIPTION | The wcswidth() function determines the number of column positions required for <i>n</i> wide-character codes (or fewer than <i>n</i> wide-character codes if a null wide-character code is encountered before <i>n</i> wide-character codes are exhausted) in the string pointed to by <i>pwcs</i> . | | | | | | |
| RETURN VALUES | The wcswidth() function either returns 0 (if <i>pwcs</i> points to a null wide-character code), or returns the number of column positions to be occupied by the wide-character string pointed to by <i>pwcs</i> , or returns -1 (if any of the first <i>n</i> wide-character codes in the wide-character string pointed to by <i>pwcs</i> is not a printing wide-character code). | | | | | | |
| ERRORS | No errors are defined. | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | | | |
| | <table border="1"> <thead> <tr> <th style="text-align: center;">ATTRIBUTE TYPE</th> <th style="text-align: center;">ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe with exceptions</td> </tr> <tr> <td>CSI</td> <td>Enabled</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe with exceptions | CSI | Enabled |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| MT-Level | MT-Safe with exceptions | | | | | | |
| CSI | Enabled | | | | | | |
| SEE ALSO | setlocale(3C) , wcwidth(3C) , attributes(5) | | | | | | |

| | |
|----------------------|--|
| NAME | wcsxfrm, wsxfrm – wide character string transformation |
| SYNOPSIS | <pre>#include <wchar.h> size_t wcsxfrm(wchar_t * ws1, const wchar_t * ws2, size_t n); size_t wsxfrm(wchar_t * ws1, const wchar_t * ws2, size_t n);</pre> |
| DESCRIPTION | <p>The wcsxfrm() and wsxfrm() functions transform the wide character string pointed to by <i>ws2</i> and place the resulting wide character string into the array pointed to by <i>ws1</i>. The transformation is such that if either the wscmp(3C) or wscmp(3C) functions are applied to two transformed wide strings, they return a value greater than, equal to, or less than 0, corresponding to the result of the wscoll(3C) or wscoll(3C) function applied to the same two original wide character strings. No more than <i>n</i> wide-character codes are placed into the resulting array pointed to by <i>ws1</i>, including the terminating null wide-character code. If <i>n</i> is 0, <i>ws1</i> is permitted to be a null pointer. If copying takes place between objects that overlap, the behavior is undefined.</p> |
| RETURN VALUES | <p>The wcsxfrm() and wsxfrm() functions return the length of the transformed wide character string (not including the terminating null wide-character code). If the value returned is <i>n</i> or more, the contents of the array pointed to by <i>ws1</i> are indeterminate.</p> <p>On error, wcsxfrm() and wsxfrm() return <code>(size_t)-1</code> and set <code>errno</code> to indicate the error.</p> |
| ERRORS | <p>The wcsxfrm() and wsxfrm() functions may fail if:</p> <p>EINVAL The wide character string pointed to by <i>ws2</i> contains wide-character codes outside the domain of the collating sequence.</p> <p>ENOSYS The function is not supported.</p> |
| USAGE | <p>The transformation function is such that two transformed wide character strings can be ordered by the wscmp() or wscmp() functions as appropriate to collating sequence information in the program's locale (category <code>LC_COLLATE</code>).</p> <p>The fact that when <i>n</i> is 0, <i>ws1</i> is permitted to be a null pointer, is useful to determine the size of the <i>ws1</i> array prior to making the transformation.</p> <p>Because no return value is reserved to indicate an error, an application wishing to check for error situations should set <code>errno</code> to 0, call wcsxfrm() or wsxfrm(), then check <code>errno</code> and if it is non-zero, assume an error has occurred.</p> |

The **wcsxfrm()** and **wsxfrm()** functions can be used safely in multithreaded applications as long as **setlocale(3C)** is not being called to change the locale.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |
| CSI | Enabled |

SEE ALSO

setlocale(3C), **wscmp(3C)**, **wscoll(3C)**, **wscmp(3C)**, **wscoll(3C)**, **attributes(5)**

| NAME | wctob – wide-character to single-byte conversion | | | | |
|----------------------|--|----------------|-----------------|----------|-------------------------|
| SYNOPSIS | <pre>#include <stdio.h> #include <wchar.h> int wctob(wint_t c);</pre> | | | | |
| DESCRIPTION | <p>The wctob() function determines whether <i>c</i> corresponds to a member of the extended character set whose character representation is a single byte when in the initial shift state.</p> <p>The behavior of this function is affected by the LC_CTYPE category of the current locale. See environ(5)</p> | | | | |
| RETURN VALUES | The wctob() function returns EOF if <i>c</i> does not correspond to a character with length one in the initial shift state. Otherwise, it returns the single-byte representation of that character. | | | | |
| ERRORS | No errors are defined. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe with exceptions</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe with exceptions |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe with exceptions | | | | |
| SEE ALSO | btowc(3C) , setlocale(3C) , attributes(5) , environ(5) | | | | |
| NOTES | The wctob() function can be used safely in multithreaded applications, as long as setlocale(3C) is not being called to change the locale. | | | | |

| NAME | wctomb – convert a wide-character code to a character | | | | | | |
|----------------------|--|----------------|-----------------|----------|-------------------------|-----|---------|
| SYNOPSIS | <pre>#include <stdlib.h> int wctomb(char *s, wchar_t wchar);</pre> | | | | | | |
| DESCRIPTION | <p>The wctomb() function determines the number of bytes needed to represent the character corresponding to the wide-character code whose value is <i>wchar</i>. It stores the character representation (possibly multiple bytes) in the array object pointed to by <i>s</i> (if <i>s</i> is not a null pointer). At most <code>MB_CUR_MAX</code> bytes are stored.</p> <p>A call with <i>s</i> as a null pointer causes this function to return 0. The behavior of this function is affected by the <code>LC_CTYPE</code> category of the current locale.</p> | | | | | | |
| RETURN VALUES | <p>If <i>s</i> is a null pointer, wctomb() returns 0 value. If <i>s</i> is not a null pointer, wctomb() returns <code>-1</code> if the value of <i>wchar</i> does not correspond to a valid character, or returns the number of bytes that constitute the character corresponding to the value of <i>wchar</i>.</p> <p>In no case will the value returned be greater than the value of the <code>MB_CUR_MAX</code> macro.</p> | | | | | | |
| ERRORS | No errors are defined. | | | | | | |
| USAGE | The wctomb() function can be used safely in a multithreaded application, as long as setlocale(3C) is not being called to change the locale. | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe with exceptions</td> </tr> <tr> <td>CSI</td> <td>Enabled</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe with exceptions | CSI | Enabled |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| MT-Level | MT-Safe with exceptions | | | | | | |
| CSI | Enabled | | | | | | |
| SEE ALSO | mblen(3C) , mbstowcs(3C) , mbtowc(3C) , setlocale(3C) , wcstombs(3C) , attributes(5) | | | | | | |

| NAME | wctrans – define character mapping | | | | | | |
|----------------------|---|----------------|-----------------|----------|-------------------------|-----|---------|
| SYNOPSIS | <pre>#include <wctype.h> wctrans_t wctrans(const char *charclass);</pre> | | | | | | |
| DESCRIPTION | <p>The wctrans() function is defined for valid character mapping names identified in the current locale. The <i>charclass</i> is a string identifying a generic character mapping name for which codeset-specific information is required. The following character mapping names are defined in all locales – "tolower" and "toupper".</p> <p>The function returns a value of type <code>wctrans_t</code>, which can be used as the second argument to subsequent calls of towctrans(3C). The wctrans() function determines values of <code>wctrans_t</code> according to the rules of the coded character set defined by character mapping information in the program's locale (category <code>LC_CTYPE</code>). The values returned by wctrans() are valid until a call to setlocale(3C) that modifies the category <code>LC_CTYPE</code>.</p> | | | | | | |
| RETURN VALUES | The wctrans() function returns 0 if the given character mapping name is not valid for the current locale (category <code>LC_CTYPE</code>), otherwise it returns a non-zero object of type <code>wctrans_t</code> that can be used in calls to towctrans(3C) . | | | | | | |
| ERRORS | <p>The wctrans() function may fail if:</p> <p>EINVAL The character mapping name pointed to by <i>charclass</i> is not valid in the current locale.</p> | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe with exceptions</td> </tr> <tr> <td>CSI</td> <td>Enabled</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe with exceptions | CSI | Enabled |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | |
| MT-Level | MT-Safe with exceptions | | | | | | |
| CSI | Enabled | | | | | | |
| SEE ALSO | setlocale(3C) , towctrans(3C) , attributes(5) | | | | | | |

| NAME | wctype – define character class | | | | | | | | | | | | |
|----------------------|---|----------------|-----------------|----------|-------------------------|-------|---------|-------|-------|-------|-------|-------|--------|
| SYNOPSIS | <pre>#include <wchar.h> wctype_t wctype(const char *charclass);</pre> | | | | | | | | | | | | |
| DESCRIPTION | <p>The wctype() function is defined for valid character class names as defined in the current locale. The <i>charclass</i> is a string identifying a generic character class for which codeset-specific type information is required. The following character class names are defined in all locales:</p> <table border="1" data-bbox="500 680 1395 854"> <tbody> <tr> <td>alnum</td> <td>alpha</td> <td>blank</td> </tr> <tr> <td>cntrl</td> <td>digit</td> <td>graph</td> </tr> <tr> <td>lower</td> <td>print</td> <td>punct</td> </tr> <tr> <td>space</td> <td>upper</td> <td>xdigit</td> </tr> </tbody> </table> <p>Additional character class names defined in the locale definition file (category LC_CTYPE) can also be specified.</p> <p>The function returns a value of type <code>wctype_t</code>, which can be used as the second argument to subsequent calls of iswctype(3C). wctype() determines values of <code>wctype_t</code> according to the rules of the coded character set defined by character type information in the program's locale (category LC_CTYPE). The values returned by wctype() are valid until a call to setlocale(3C) that modifies the category LC_CTYPE.</p> | alnum | alpha | blank | cntrl | digit | graph | lower | print | punct | space | upper | xdigit |
| alnum | alpha | blank | | | | | | | | | | | |
| cntrl | digit | graph | | | | | | | | | | | |
| lower | print | punct | | | | | | | | | | | |
| space | upper | xdigit | | | | | | | | | | | |
| RETURN VALUES | The wctype() function returns 0 if the given character class name is not valid for the current locale (category LC_CTYPE); otherwise it returns an object of type <code>wctype_t</code> that can be used in calls to iswctype() . | | | | | | | | | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | | | | | | | | | |
| | <table border="1" data-bbox="500 1367 1395 1495"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe with exceptions</td> </tr> <tr> <td>CSI</td> <td>Enabled</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe with exceptions | CSI | Enabled | | | | | | |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | | | | | | | | | |
| MT-Level | MT-Safe with exceptions | | | | | | | | | | | | |
| CSI | Enabled | | | | | | | | | | | | |
| SEE ALSO | iswctype(3C) , setlocale(3C) , attributes(5) | | | | | | | | | | | | |

NAME | `wcwidth` – number of column positions of a wide-character code

SYNOPSIS | `#include <wctype.h>`

| `int wcwidth(wchar_t wc);`

DESCRIPTION | The `wcwidth()` function determines the number of column positions required for the wide character `wc`. The value of `wc` must be a character representable as a `wchar_t`, and must be a wide-character code corresponding to a valid character in the current locale.

RETURN VALUES | The `wcwidth()` function either returns 0 (if `wc` is a null wide-character code), or returns the number of column positions to be occupied by the wide-character code `wc`, or returns -1 (if `wc` does not correspond to a printing wide-character code).

ERRORS | No errors are defined.

ATTRIBUTES | See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-------------------------|
| MT-Level | MT-Safe with exceptions |
| CSI | Enabled |

SEE ALSO | `setlocale(3C)`, `wcswidth(3C)`, `attributes(5)`

NAME wmemchr – find a wide-character in memory

SYNOPSIS #include <wchar.h>

```
wchar_t *wmemchr(const wchar_t *ws, wchar_t wc, size_t n);
```

DESCRIPTION The **wmemchr()** function locates the first occurrence of *wc* in the initial *n* wide-characters of the object pointed to be *ws*. This function is not affected by locale and all *wchar_t* values are treated identically. The null wide-character and *wchar_t* values not corresponding to valid characters are not treated specially.

If *n* is 0, *ws* must be a valid pointer and the function behaves as if no valid occurrence of *wc* is found.

RETURN VALUES The **wmemchr()** function returns a pointer to the located wide-character, or a null pointer if the wide-character does not occur in the object.

ERRORS No errors are defined.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **wmemcmp(3C)**, **wmemcpy(3C)**, **wmemmove(3C)**, **wmemset(3C)**, **attributes(5)**

NAME | wmemcmp – compare wide-characters in memory

SYNOPSIS | #include <wchar.h>

| int wmemcmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);

DESCRIPTION | The **wmemcmp()** function compares the first *n* wide-characters of the object pointed to by *ws1* to the first *n* wide-characters of the object pointed to by *ws2*. This function is not affected by locale and all `wchar_t` values are treated identically. The null wide-character and `wchar_t` values not corresponding to valid characters are not treated specially.

| If *n* is zero, *ws1* and *ws2* must be a valid pointers and the function behaves as if the two objects compare equal.

RETURN VALUES | The **wmemcmp()** function returns an integer greater than, equal to, or less than 0, accordingly as the object pointed to by *ws1* is greater than, equal to, or less than the object pointed to by *ws2*.

ERRORS | No errors are defined.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **wmemcmp(3C)**, **wmemcpy(3C)**, **wmemmove(3C)**, **wmemset(3C)**, **attributes(5)**

NAME wmemcpy – copy wide-characters in memory

SYNOPSIS #include <wchar.h>

```
wchar_t *wmemcpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
```

DESCRIPTION The **wmemcpy()** function copies *n* wide-characters from the object pointed to by *ws2* to the object pointed to be *ws1*. This function is not affected by locale and all `wchar_t` values are treated identically. The null wide-character and `wchar_t` values not corresponding to valid characters are not treated specially.

If *n* is zero, *ws1* and *ws2* must be a valid pointers, and the function copies zero wide-characters.

RETURN VALUES The **wmemcpy()** function returns the value of *ws1*.

ERRORS No errors are defined.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **wmemchr(3C)**, **wmemcmp(3C)**, **wmemmove(3C)**, **wmemset(3C)**, **attributes(5)**

NAME | wmemmove – copy wide-characters in memory with overlapping areas

SYNOPSIS | #include <wchar.h>

| wchar_t *wmemmove(wchar_t *ws1, const wchar_t *ws2, size_t n);

DESCRIPTION | The **wmemmove()** function copies *n* wide-characters from the object pointed to by *ws2* to the object pointed to by *ws1*. Copying takes place as if the *n* wide-characters from the object pointed to by *ws2* are first copied into a temporary array of *n* wide-characters that does not overlap the objects pointed to by *ws1* or *ws2*, and then the *n* wide-characters from the temporary array are copied into the object pointed to by *ws1*.

| This function is not affected by locale and all `wchar_t` values are treated identically. The null wide-character and `wchar_t` values not corresponding to valid characters are not treated specially.

| If *n* is 0, *ws1* and *ws2* must be a valid pointers, and the function copies zero wide-characters.

RETURN VALUES | The **wmemmove()** function returns the value of *ws1*.

ERRORS | No errors are defined.

ATTRIBUTES | See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO | **wmemchr(3C)**, **wmemcmp(3C)**, **wmemcpy(3C)**, **wmemset(3C)**, **attributes(5)**

| NAME | wmemset – set wide-characters in memory | | | | |
|----------------------|--|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <wchar.h> wchar_t *wmemset(wchar_t *ws, wchar_t wc, size_t n);</pre> | | | | |
| DESCRIPTION | <p>The wmemset() function copies the value of <i>wc</i> into each of the first <i>n</i> wide-characters of the object pointed to by <i>ws</i>. This function is not affected by locale and all <i>wchar_t</i> values are treated identically. The null wide-character and <i>wchar_t</i> values not corresponding to valid characters are not treated specially.</p> <p>If <i>n</i> is 0, <i>ws</i> must be a valid pointer and the function copies zero wide-characters.</p> | | | | |
| RETURN VALUES | The wmemset() functions returns the value of <i>ws</i> . | | | | |
| ERRORS | No errors are defined. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | wmemchr(3C) , wmemcmp(3C) , wmemcpy(3C) , wmemmove(3C) , attributes(5) | | | | |

| | |
|---------------------------------|--|
| NAME | wordexp, wordfree – perform word expansions |
| SYNOPSIS | <pre>#include <wordexp.h> int wordexp(const char * words, wordexp_t * pwordexp, int flags); void wordfree(wordexp_t * pwordexp);</pre> |
| DESCRIPTION | <p>The wordexp() function performs word expansions, subject to quoting, and places the list of expanded words into the structure pointed to by <i>pwordexp</i> .</p> <p>The wordfree() function frees any memory allocated by wordexp() associated with <i>pwordexp</i> .</p> |
| <i>words</i> Argument | <p>The <i>words</i> argument is a pointer to a string containing one or more words to be expanded. The expansions will be the same as would be performed by the shell if <i>words</i> were the part of a command line representing the arguments to a utility. Therefore, <i>words</i> must not contain an unquoted NEWLINE or any of the unquoted shell special characters:</p> <pre> & ; < ></pre> <p>except in the context of command substitution. It also must not contain unquoted parentheses or braces, except in the context of command or variable substitution. If the argument <i>words</i> contains an unquoted comment character (number sign) that is the beginning of a token, wordexp() may treat the comment character as a regular character, or may interpret it as a comment indicator and ignore the remainder of <i>words</i> .</p> |
| <i>pwordexp</i> Argument | <p>The structure type <code>wordexp_t</code> is defined in the header <code><wordexp.h></code> and includes at least the following members:</p> <pre>size_t we_wordc Count of words matched by <i>words</i> . char **we_wordv Pointer to list of expanded words. size_t we_offs Slots to reserve at the beginning of <i>pwordexp</i>-> we_wordv .</pre> <p>The wordexp() function stores the number of generated words into <i>pwordexp</i>-><code>we_wordc</code> and a pointer to a list of pointers to words in <i>pwordexp</i>-><code>we_wordv</code> . Each individual field created during field splitting is a separate word in the <i>pwordexp</i>-><code>we_wordv</code> list. The words are in order. The first pointer after the last word pointer will be a null pointer.</p> <p>It is the caller's responsibility to allocate the storage pointed to by <i>pwordexp</i> . The wordexp() function allocates other space as needed, including memory</p> |

pointed to by *pwordexp*→we_wordv . The **wordfree()** function frees any memory associated with *pwordexp* from a previous call to **wordexp()** .

flags Argument

The *flags* argument is used to control the behavior of **wordexp()** . The value of *flags* is the bitwise inclusive OR of zero or more of the following constants, which are defined in <wordexp.h> :

| | |
|--------------|--|
| WRDE_APPEND | Append words generated to the ones from a previous call to wordexp() . |
| WRDE_DOOFFS | Make use of <i>pwordexp</i> →we_offs . If this flag is set, <i>pwordexp</i> →we_offs is used to specify how many NULL pointers to add to the beginning of <i>pwordexp</i> →we_wordv . In other words, <i>pwordexp</i> →we_wordv will point to <i>pwordexp</i> →we_offs NULL pointers, followed by <i>pwordexp</i> →we_wordc word pointers, followed by a NULL pointer. |
| WRDE_NOCMD | Fail if command substitution is requested. |
| WRDE_REUSE | The <i>pwordexp</i> argument was passed to a previous successful call to wordexp() , and has not been passed to wordfree() . The result will be the same as if the application had called wordfree() and then called wordexp() without WRDE_REUSE . |
| WRDE_SHOWERR | Do not redirect stderr to /dev/null . |
| WRDE_UNDEF | Report error on an attempt to expand an undefined shell variable. |

The WRDE_APPEND flag can be used to append a new set of words to those generated by a previous call to **wordexp()** . The following rules apply when two or more calls to **wordexp()** are made with the same value of *pwordexp* and without intervening calls to **wordfree()** :

1. The first such call must not set WRDE_APPEND . All subsequent calls must set it.
2. All of the calls must set WRDE_DOOFFS , or all must not set it.
3. After the second and each subsequent call, *pwordexp*→we_wordv will point to a list containing the following:
 - a. zero or more NULL pointers, as specified by WRDE_DOOFFS and *pwordexp*→we_offs .
 - b. pointers to the words that were in the *pwordexp*→we_wordv list before the call, in the same order as before.

- c. pointers to the new words generated by the latest call, in the specified order.
- 4. The count returned in *pwordexp*→ *we_wordc* will be the total number of words from all of the calls.
- 5. The application can change any of the fields after a call to **wordexp()** , but if it does it must reset them to the original value before a subsequent call, using the same *pwordexp* value, to **wordfree()** or **wordexp()** with the *WRDE_APPEND* or *WRDE_REUSE* flag.

If *words* contains an unquoted:

```
NEWLINE | & ; < > ( ) { }
```

in an inappropriate context, **wordexp()** will fail, and the number of expanded words will be zero.

Unless *WRDE_SHOWERR* is set in *flags* , **wordexp()** will redirect *stderr* to */dev/null* for any utilities executed as a result of command substitution while expanding *words* .

If *WRDE_SHOWERR* is set, **wordexp()** may write messages to *stderr* if syntax errors are detected while expanding *words* . If *WRDE_DOOFFS* is set, then *pwordexp*→ *we_offs* must have the same value for each **wordexp()** call and **wordfree()** call using a given *pwordexp* .

The following constants are defined as error return values:

WRDE_BADCHAR One of the unquoted characters:

```
NEWLINE | & ; < > ( ) { }
```

appears in *words* in an inappropriate context.

WRDE_BADVAL Reference to undefined shell variable when *WRDE_UNDEF* is set in *flags* .

WRDE_CMDSUB Command substitution requested when *WRDE_NOCMD* was set in *flags*.

WRDE_NOSPACE Attempt to allocate memory failed.

WRDE_SYNTAX Shell syntax error, such as unbalanced parentheses or unterminated string.

RETURN VALUES

On successful completion, **wordexp()** returns 0 .

Otherwise, a non-zero value as described in `<wordexp.h>` is returned to indicate an error. If **wordexp()** returns the value `WRDE_NOSPACE` , then *pwordexp*-> `we_wordc` and *pwordexp*-> `we_wordv` will be updated to reflect any words that were successfully expanded. In other cases, they will not be modified.

The **wordfree()** function returns no value.

ERRORS

No errors are defined.

USAGE

This function is intended to be used by an application that wants to do all of the shell's expansions on a word or words obtained from a user. For example, if the application prompts for a filename (or list of filenames) and then uses **wordexp()** to process the input, the user could respond with anything that would be valid as input to the shell.

The `WRDE_NOCMD` flag is provided for applications that, for security or other reasons, want to prevent a user from executing shell command. Disallowing unquoted shell special characters also prevents unwanted side effects such as executing a command or writing a file.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

fnmatch(3C) , **glob(3C)** , **attributes(5)**

NAME wprintf – formatted output conversion

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

int wprintf(wchar_t *s, const char *format, /* arg */ ... );
```

DESCRIPTION The **wprintf()** function outputs a Process Code string ending with a Process Code (`wchar_t`) null character. It is the user's responsibility to allocate enough space for this `wchar_t` string.

This returns the number of Process Code characters (excluding the null terminator) that have been written. The conversion specifications and behavior of **wprintf()** are the same as the regular **sprintf(3S)** function except that the result is a Process Code string for `wprintf()`, and on Extended Unix Code (EUC) character string for **sprintf()**.

RETURN VALUES Upon successful completion, **wprintf()** returns the number of characters printed. Otherwise, a negative value is returned.

ATTRIBUTES See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **wscanf(3C)**, **printf(3S)**, **scanf(3S)**, **sprintf(3S)**, **attributes(5)**

NAME wscanf – formatted input conversion

SYNOPSIS

```
#include<stdio.h>
#include <wchar.h>

int wscanf(wchar_t *s, const char *format, /* pointer */ ... );
```

DESCRIPTION

The **wscanf()** function reads Process Code characters from the Process Code string *s*, interprets them according to the *format*, and stores the results in its arguments. It expects, as arguments, a control string *format*, and a set of *pointer* arguments indicating where the converted input should be stored. The results are undefined if there are insufficient *args* for the format. If the format is exhausted while *args* remain, the excess *args* are simply ignored.

The conversion specifications and behavior of **wscanf()** are the same as the regular **scanf(3S)** function except that the source is a Process Code string for **wscanf()** and on Extended Unix Code (EUC) character string for **scanf(3S)**.

RETURN VALUES

Upon successful completion, **wscanf()** returns the number of characters matched. Otherwise, it returns a negative value.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO **wsprintf(3C)**, **printf(3S)**, **scanf(3S)**, **attributes(5)**

| NAME | wstring, wscasecmp, wscasecmp, wsdup, wscol – Process Code string operations | | | | |
|--------------------|---|----------------|-----------------|----------|---------|
| SYNOPSIS | <pre>#include <wchar.h> int wscasecmp(const wchar_t * s1, const wchar_t * s2); int wscasecmp(const wchar_t * s1, const wchar_t * s2, int n); wchar_t * wsdup(const wchar_t * s); int wscol(const wchar_t * s);</pre> | | | | |
| DESCRIPTION | <p>These functions operate on Process Code strings terminated by <code>wchar_t</code> null characters. During appending or copying, these routines do not check for an overflow condition of the receiving string. In the following, <code>s</code>, <code>s1</code>, and <code>s2</code> point to Process Code strings terminated by a <code>wchar_t</code> null.</p> <p>wscasecmp(), wscasecmp() The wscasecmp() function compares its arguments, ignoring case, and returns an integer greater than, equal to, or less than 0, depending upon whether <code>s1</code> is lexicographically greater than, equal to, or less than <code>s2</code>. It makes the same comparison but compares at most <code>n</code> Process Code characters. The four Extended Unix Code (EUC) codesets are ordered from lowest to highest as 0, 2, 3, 1 when characters from different codesets are compared.</p> <p>wsdup() The wsdup() function returns a pointer to a new Process Code string, which is a duplicate of the string pointed to by <code>s</code>. The space for the new string is obtained using <code>malloc(3C)</code>. If the new string cannot be created, a null pointer is returned.</p> <p>wscol() The wscol() function returns the screen display width (in columns) of the Process Code string <code>s</code>.</p> | | | | |
| ATTRIBUTES | <p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>MT-Safe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | MT-Safe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | MT-Safe | | | | |
| SEE ALSO | <code>malloc(3C)</code> , <code>string(3C)</code> , <code>wcstring(3C)</code> , <code>attributes(5)</code> | | | | |

| | |
|----------------------|---|
| NAME | wunctrl – convert a wide character to printable form |
| SYNOPSIS | <pre>#include <curses.h> wchar_t *wunctrl(cchar_t wc);</pre> |
| PARAMETERS | <p><code>wc</code> Is a wide character.</p> |
| DESCRIPTION | <p>The wunctrl() function converts the wide character code <code>wc</code> into a printable form (if unprintable). Control characters are displayed using the <code>^x</code> notation where <code>^</code> identifies the control key and <code>x</code> represents an alphanumeric character that is pressed while the control key is held down.</p> <p>Characters which have their eighth bit set are represented using the meta notation <code>M-X</code> where <code>X</code> is the byte with eighth bit stripped. This stripped byte will represent either a printable character or a control character. If it is a control character, <code>X</code> is actually represented using <code>^X</code> notation. For example, <code>0xCD</code> in ASCII is <code>M-^K</code>.</p> |
| RETURN VALUES | On success, the wunctrl() function returns the generated string. Otherwise, it returns a null pointer. |
| ERRORS | None. |
| SEE ALSO | keyname(3XC) , unctrl(3XC) |

| NAME | xdr – library routines for external data representation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------------|---|-------------|-----------------------|-----------|------------------------|----------|-----------------------|-----------|------------------------|----------|-----------------------|-------------|----------------------|-------------|-----------------------|------------|-----------------------|----------|-----------------------|-----------|-----------------------|----------|-----------------------|------------|----------------------|-----------|-----------------------|------------|----------------------|---------|-----------------------|----------|-----------------------|----------------|-----------------------|------------|------------------------|-------------|------------------------|---------------|-----------------------|
| DESCRIPTION | XDR routines allow C programmers to describe arbitrary data structures in a machine-independent fashion. Data for remote procedure calls (RPC) are transmitted using these routines. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Index to Routines | <p>The following table lists XDR routines and the manual reference pages on which they are described:</p> <table border="0"> <thead> <tr> <th style="text-align: left;">XDR Routine</th> <th style="text-align: left;">Manual Reference Page</th> </tr> </thead> <tbody> <tr><td>xdr_array</td><td>xdr_complex(3N)</td></tr> <tr><td>xdr_bool</td><td>xdr_simple(3N)</td></tr> <tr><td>xdr_bytes</td><td>xdr_complex(3N)</td></tr> <tr><td>xdr_char</td><td>xdr_simple(3N)</td></tr> <tr><td>xdr_control</td><td>xdr_admin(3N)</td></tr> <tr><td>xdr_destroy</td><td>xdr_create(3N)</td></tr> <tr><td>xdr_double</td><td>xdr_simple(3N)</td></tr> <tr><td>xdr_enum</td><td>xdr_simple(3N)</td></tr> <tr><td>xdr_float</td><td>xdr_simple(3N)</td></tr> <tr><td>xdr_free</td><td>xdr_simple(3N)</td></tr> <tr><td>xdr_getpos</td><td>xdr_admin(3N)</td></tr> <tr><td>xdr_hyper</td><td>xdr_simple(3N)</td></tr> <tr><td>xdr_inline</td><td>xdr_admin(3N)</td></tr> <tr><td>xdr_int</td><td>xdr_simple(3N)</td></tr> <tr><td>xdr_long</td><td>xdr_simple(3N)</td></tr> <tr><td>xdr_longlong_t</td><td>xdr_simple(3N)</td></tr> <tr><td>xdr_opaque</td><td>xdr_complex(3N)</td></tr> <tr><td>xdr_pointer</td><td>xdr_complex(3N)</td></tr> <tr><td>xdr_quadruple</td><td>xdr_simple(3N)</td></tr> </tbody> </table> | XDR Routine | Manual Reference Page | xdr_array | xdr_complex(3N) | xdr_bool | xdr_simple(3N) | xdr_bytes | xdr_complex(3N) | xdr_char | xdr_simple(3N) | xdr_control | xdr_admin(3N) | xdr_destroy | xdr_create(3N) | xdr_double | xdr_simple(3N) | xdr_enum | xdr_simple(3N) | xdr_float | xdr_simple(3N) | xdr_free | xdr_simple(3N) | xdr_getpos | xdr_admin(3N) | xdr_hyper | xdr_simple(3N) | xdr_inline | xdr_admin(3N) | xdr_int | xdr_simple(3N) | xdr_long | xdr_simple(3N) | xdr_longlong_t | xdr_simple(3N) | xdr_opaque | xdr_complex(3N) | xdr_pointer | xdr_complex(3N) | xdr_quadruple | xdr_simple(3N) |
| XDR Routine | Manual Reference Page | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| xdr_array | xdr_complex(3N) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| xdr_bool | xdr_simple(3N) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| xdr_bytes | xdr_complex(3N) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| xdr_char | xdr_simple(3N) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| xdr_control | xdr_admin(3N) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| xdr_destroy | xdr_create(3N) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| xdr_double | xdr_simple(3N) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| xdr_enum | xdr_simple(3N) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| xdr_float | xdr_simple(3N) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| xdr_free | xdr_simple(3N) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| xdr_getpos | xdr_admin(3N) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| xdr_hyper | xdr_simple(3N) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| xdr_inline | xdr_admin(3N) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| xdr_int | xdr_simple(3N) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| xdr_long | xdr_simple(3N) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| xdr_longlong_t | xdr_simple(3N) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| xdr_opaque | xdr_complex(3N) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| xdr_pointer | xdr_complex(3N) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| xdr_quadruple | xdr_simple(3N) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| | |
|--------------------|-----------------|
| xdr_reference | xdr_complex(3N) |
| xdr_setpos | xdr_admin(3N) |
| xdr_short | xdr_simple(3N) |
| xdr_sizeof | xdr_admin(3N) |
| xdr_string | xdr_complex(3N) |
| xdr_u_char | xdr_simple(3N) |
| xdr_u_hyper | xdr_simple(3N) |
| xdr_u_int | xdr_simple(3N) |
| xdr_u_long | xdr_simple(3N) |
| xdr_u_longlong_t | xdr_simple(3N) |
| xdr_u_short | xdr_simple(3N) |
| xdr_union | xdr_complex(3N) |
| xdr_vector | xdr_complex(3N) |
| xdr_void | xdr_simple(3N) |
| xdr_wrapstring | xdr_complex(3N) |
| xdrmem_create | xdr_create(3N) |
| xdrrec_create | xdr_create(3N) |
| xdrrec_endofrecord | xdr_admin(3N) |
| xdrrec_eof | xdr_admin(3N) |
| xdrrec_readbytes | xdr_admin(3N) |
| xdrrec_skiprecord | xdr_admin(3N) |
| xdrstdio_create | xdr_create(3N) |

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

`rpc(3N)`, `xdr_admin(3N)`, `xdr_complex(3N)`, `xdr_create(3N)`,
`xdr_simple(3N)`, `attributes(5)`

| | |
|--------------------|--|
| NAME | xdr_admin, xdr_control, xdr_getpos, xdr_inline, xdrrec_endofrecord, xdrrec_eof, xdrrec_readbytes, xdrrec_skiprecord, xdr_setpos, xdr_sizeof – library routines for external data representation |
| DESCRIPTION | <p>XDR library routines allow C programmers to describe arbitrary data structures in a machine-independent fashion. Protocols such as remote procedure calls (RPC) use these routines to describe the format of the data.</p> <p>These routines deal specifically with the management of the XDR stream.</p> |
| Routines | <p>See rpc(3N) for the definition of the XDR data structure. Note that any buffers passed to the XDR routines must be properly aligned. It is suggested either that malloc(3C) be used to allocate these buffers, or that the programmer insure that the buffer address is divisible evenly by four.</p> <pre>#include <rpc/xdr.h> bool_t xdr_control(XDR * <i>xdrs</i> , int <i>req</i> , void * <i>info</i>);</pre> <p>A function macro to change or retrieve various information about an XDR stream. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i> is XDR_GET_BYTES_AVAIL and its argument type is xdr_bytesrec *. They return the number of bytes left unconsumed in the stream and a flag indicating whether or not this is the last fragment.</p> <pre>uint_t xdr_getpos(const XDR * <i>xdrs</i>);</pre> <p>A macro that invokes the get-position routine associated with the XDR stream, <i>xdrs</i>. The routine returns an unsigned integer, which indicates the position of the XDR byte stream. A desirable feature of XDR streams is that simple arithmetic works with this number, although the XDR stream instances need not guarantee this. Therefore, applications written for portability should not depend on this feature.</p> <pre>long *xdr_inline(XDR * <i>xdrs</i> , const int <i>len</i>);</pre> <p>A macro that invokes the in-line routine associated with the XDR stream, <i>xdrs</i>. The routine returns a pointer to a contiguous piece of the stream's buffer; <i>len</i> is the byte length of the desired buffer. Note: pointer is cast to long *.</p> <p>Warning: xdr_inline() may return NULL (0) if it cannot allocate a contiguous piece of a buffer. Therefore the behavior may vary among stream instances; it exists for the sake of efficiency, and applications written for portability should not depend on this feature.</p> |

```
bool_t xdrrec_endofrecord(XDR *xdrs, int sendnow );
```

This routine can be invoked only on streams created by **xdrrec_create()** . See **xdr_create(3N)** . The data in the output buffer is marked as a completed record, and the output buffer is optionally written out if *sendnow* is non-zero. This routine returns **TRUE** if it succeeds, **FALSE** otherwise.

```
bool_t xdrrec_eof(XDR * xdrs );
```

This routine can be invoked only on streams created by **xdrrec_create()** . After consuming the rest of the current record in the stream, this routine returns **TRUE** if there is no more data in the stream's input buffer. It returns **FALSE** if there is additional data in the stream's input buffer.

```
int xdrrec_readbytes(XDR * xdrs , caddr_t addr , uint_t nbytes );
```

This routine can be invoked only on streams created by **xdrrec_create()** . It attempts to read *nbytes* bytes from the XDR stream into the buffer pointed to by *addr* . Upon success this routine returns the number of bytes read. Upon failure, it returns **-1** . A return value of **0** indicates an end of record.

```
bool_t xdrrec_skiprecord(XDR * xdrs );
```

This routine can be invoked only on streams created by **xdrrec_create()** . See **xdr_create(3N)** . It tells the XDR implementation that the rest of the current record in the stream's input buffer should be discarded. This routine returns **TRUE** if it succeeds, **FALSE** otherwise.

```
bool_t xdr_setpos(XDR * xdrs , const uint_t pos );
```

A macro that invokes the set position routine associated with the XDR stream *xdrs* . The parameter *pos* is a position value obtained from **xdr_getpos()** . This routine returns **TRUE** if the XDR stream was repositioned, and **FALSE** otherwise.

Warning: it is difficult to reposition some types of XDR streams, so this routine may fail with one type of stream and succeed with another. Therefore, applications written for portability should not depend on this feature.

```
unsigned long xdr_sizeof(xdrproc_t func , void * data );
```

This routine returns the number of bytes required to encode *data* using the XDR filter function *func*, excluding potential overhead such as RPC headers or record markers. 0 is returned on error. This information might be used to select between transport protocols, or to determine the buffer size for various lower levels of RPC client and server creation routines, or to allocate storage when XDR is used outside of the RPC subsystem.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

`malloc(3C)`, `rpc(3N)`, `xdr_complex(3N)`, `xdr_create(3N)`,
`xdr_simple(3N)`, `attributes(5)`

NAME xdr_complex, xdr_array, xdr_bytes, xdr_opaque, xdr_pointer, xdr_reference, xdr_string, xdr_union, xdr_vector, xdr_wrapstring – library routines for external data representation

DESCRIPTION XDR library routines allow C programmers to describe complex data structures in a machine-independent fashion. Protocols such as remote procedure calls (RPC) use these routines to describe the format of the data. These routines are the XDR library routines for complex data structures. They require the creation of XDR streams. See `xdr_create(3N)`.

Routines See `rpc(3N)` for the definition of the XDR data structure. Note that any buffers passed to the XDR routines must be properly aligned. It is suggested either that `malloc()` be used to allocate these buffers, or that the programmer insure that the buffer address is divisible evenly by four.

```
#include <rpc/xdr.h>
```

```
bool_t xdr_array(XDR * xdrs , caddr_t * arrip , uint_t * sizep , const uint_t  
maxsize , const uint_t elsize , const xdrproc_t elproc );
```

`xdr_array()` translates between variable-length arrays and their corresponding external representations. The parameter `arrip` is the address of the pointer to the array, while `sizep` is the address of the element count of the array; this element count cannot exceed `maxsize`. The parameter `elsize` is the size of each of the array's elements, and `elproc` is an XDR routine that translates between the array elements' C form and their external representation. If `* arrip` is `NULL` when decoding, `xdr_array()` allocates memory and `* arrip` points to it. This routine returns `TRUE` if it succeeds, `FALSE` otherwise.

```
bool_t xdr_bytes(XDR * xdrs , char ** sp , uint_t * sizep , const uint_t  
maxsize );
```

`xdr_bytes()` translates between counted byte strings and their external representations. The parameter `sp` is the address of the string pointer. The length of the string is located at address `sizep`; strings cannot be longer than `maxsize`. If `* sp` is `NULL` when decoding, `xdr_bytes()` allocates memory and `* sp` points to it. This routine returns `TRUE` if it succeeds, `FALSE` otherwise.

```
bool_t xdr_opaque(XDR * xdrs , caddr_t cp , const uint_t cnt );
```

`xdr_opaque()` translates between fixed size opaque data and its external representation. The parameter `cp` is the address of the opaque object, and `cnt` is its size in bytes. This routine returns `TRUE` if it succeeds, `FALSE` otherwise.

```
bool_t xdr_pointer(XDR * xdrs , char **objpp, uint_t objsize , const  
xdrproc_t xdrobj );
```

Like **xdr_reference()** except that it serializes null pointers, whereas **xdr_reference()** does not. Thus, **xdr_pointer()** can represent recursive data structures, such as binary trees or linked lists. If * *objpp* is NULL when decoding, **xdr_pointer()** allocates memory and * *objpp* points to it.

```
bool_t xdr_reference(XDR * xdrs , caddr_t * pp , uint_t size , const xdrproc_t proc );
```

xdr_reference() provides pointer chasing within structures. The parameter *pp* is the address of the pointer; *size* is the `sizeof` of the structure that **pp* points to; and *proc* is an XDR procedure that translates the structure between its C form and its external representation. If * *pp* is NULL when decoding, **xdr_reference()** allocates memory and * *pp* points to it. This routine returns 1 if it succeeds, 0 otherwise.

Warning: this routine does not understand null pointers. Use **xdr_pointer()** instead.

```
bool_t xdr_string(XDR * xdrs , char ** sp , const uint_t maxsize );
```

xdr_string() translates between C strings and their corresponding external representations. Strings cannot be longer than *maxsize*. Note: *sp* is the address of the string's pointer. If * *sp* is NULL when decoding, **xdr_string()** allocates memory and * *sp* points to it. This routine returns TRUE if it succeeds, FALSE otherwise. Note: **xdr_string()** can be used to send an empty string (""), but not a null string.

```
bool_t xdr_union(XDR * xdrs , enum_t * dscmp , char * unp , const struct xdr_discrim * choices , const xdrproc_t (* defaultarm ));
```

xdr_union() translates between a discriminated C union and its corresponding external representation. It first translates the discriminant of the union located at *dscmp*. This discriminant is always an `enum_t`. Next the union located at *unp* is translated. The parameter *choices* is a pointer to an array of `xdr_discrim` structures. Each structure contains an ordered pair of [*value*, *proc*]. If the union's discriminant is equal to the associated *value*, then the *proc* is called to translate the union. The end of the `xdr_discrim` structure array is denoted by a routine of value NULL. If the discriminant is not found in the *choices* array, then the *defaultarm* procedure is called (if it is not NULL). It returns TRUE if it succeeds, FALSE otherwise.

```
bool_t xdr_vector(XDR * xdrs , char * arp , const uint_t size , const uint_t elsize , const xdrproc_t elproc );
```

xdr_vector() translates between fixed-length arrays and their corresponding external representations. The parameter *arrp* is the address of the pointer to the array, while *size* is the element count of the array. The parameter *elsize* is the `sizeof` each of the array's elements, and *elproc* is an XDR routine that translates between the array elements' C form and their external representation. This routine returns `TRUE` if it succeeds, `FALSE` otherwise.

bool_t xdr_wrapstring(XDR * xdrs , char ** sp);

A routine that calls `xdr_string(xdrs , sp , maxuint)`; where *maxuint* is the maximum value of an unsigned integer.

Many routines, such as **xdr_array()** , **xdr_pointer()** , and **xdr_vector()** take a function pointer of type **xdrproc_t()** , which takes two arguments.

xdr_string() , one of the most frequently used routines, requires three arguments, while **xdr_wrapstring()** only requires two. For these routines, **xdr_wrapstring()** is desirable. This routine returns `TRUE` if it succeeds, `FALSE` otherwise.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

malloc(3C) , **rpc(3N)** , **xdr_admin(3N)** , **xdr_create(3N)** , **xdr_simple(3N)** , **attributes(5)**

| | |
|--------------------|--|
| NAME | xdr_create, xdr_destroy, xdrmem_create, xdrrec_create, xdrstdio_create – library routines for external data representation stream creation |
| DESCRIPTION | <p>XDR library routines allow C programmers to describe arbitrary data structures in a machine-independent fashion. Protocols such as remote procedure calls (RPC) use these routines to describe the format of the data.</p> <p>These routines deal with the creation of XDR streams. XDR streams have to be created before any data can be translated into XDR format.</p> |
| Routines | <p>See rpc(3N) for the definition of the <code>XDR</code>, <code>CLIENT</code>, and <code>SVCXPRT</code> data structures. Note that any buffers passed to the XDR routines must be properly aligned. It is suggested that <code>malloc(3C)</code> be used to allocate these buffers or that the programmer insure that the buffer address is divisible evenly by four.</p> <pre>#include <rpc/xdr.h> void xdr_destroy(XDR * xdrs);</pre> <p>A macro that invokes the destroy routine associated with the XDR stream, <code>xdrs</code>. Destruction usually involves freeing private data structures associated with the stream. Using <code>xdrs</code> after invoking <code>xdr_destroy()</code> is undefined.</p> <pre>void xdrmem_create(XDR * xdrs , const caddr_t addr , const uint_t size , const enum xdr_op op);</pre> <p>This routine initializes the XDR stream object pointed to by <code>xdrs</code>. The stream's data is written to, or read from, a chunk of memory at location <code>addr</code> whose length is no less than <code>size</code> bytes long. The <code>op</code> determines the direction of the XDR stream (either <code>XDR_ENCODE</code>, <code>XDR_DECODE</code>, or <code>XDR_FREE</code>).</p> <pre>void xdrrec_create(XDR * xdrs , const uint_t sendsz , const uint_t recvsz , const caddr_t handle , const int (* readit)(const void * read_handle , char * buf , const int len), const int (* writeit)(const void * write_handle , const char * buf , const int len));</pre> <p>This routine initializes the read-oriented XDR stream object pointed to by <code>xdrs</code>. The stream's data is written to a buffer of size <code>sendsz</code>; a value of 0 indicates the system should use a suitable default. The stream's data is read from a buffer of size <code>recvsz</code>; it too can be set to a suitable default by passing a 0 value. When a stream's output buffer is full, <code>writeit</code> is called. Similarly, when a stream's input buffer is empty, <code>readit</code> is called. The behavior of these two routines is similar to the system calls <code>read()</code> and <code>write()</code> (see <code>read(2)</code> and <code>write(2)</code>, respectively), except that an appropriate handle (<code>read_handle</code> or <code>write_handle</code>) is passed to the former routines as the first parameter</p> |

instead of a file descriptor. Note: the XDR stream's *op* field must be set by the caller.

Warning: this XDR stream implements an intermediate record stream. Therefore there are additional bytes in the stream to provide record boundary information.

void xdrstdio_create(XDR * *xdrs* , FILE * *file* , const enum xdr_op *op*);

This routine initializes the XDR stream object pointed to by *xdrs* . The XDR stream data is written to, or read from, the standard I/O stream *file* . The parameter *op* determines the direction of the XDR stream (either XDR_ENCODE , XDR_DECODE , or XDR_FREE).

Warning: the destroy routine associated with such XDR streams calls **fflush()** on the *file* stream, but never **fclose()** (see **fclose(3S)**).

Failure of any of these functions can be detected by first initializing the *x_ops* field in the XDR structure (*xdrs* \Rightarrow *x_ops*) to NULL before calling the *xdr*_create()* function. After the return from the *xdr*_create()* function, if the *x_ops* field is still NULL , the call has failed. If the *x_ops* field contains some other value, the call can be assumed to have succeeded.

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

read(2) , **write(2)** , **fclose(3S)** , **malloc(3C)** , **rpc(3N)** , **xdr_admin(3N)** , **xdr_complex(3N)** , **xdr_simple(3N)** , **attributes(5)**

| | |
|--------------------|--|
| NAME | xfn – overview of the XFN interface |
| DESCRIPTION | <p>The primary service provided by a federated naming system is to map a <i>composite name</i> to a <i>reference</i>. A composite name is composed of name components from one or more naming systems. A reference consists of one or more communication end points. An additional service provided by a federated naming system is to provide access to attributes associated with named objects. This extension is to satisfy most applications' additional naming service needs without cluttering the basic naming service model. XFN is a programming interface for a federated naming service.</p> <p>To use the XFN interface, include the <code>xfn/xfn.h</code> header file and link the application with <code>-lxfn</code>.</p> <p>The <code>xfn/xfn.h</code> header file contains the interface declarations for:</p> <ul style="list-style-type: none"> ■ the XFN base context interface, ■ the XFN base attribute interface, ■ status object and status codes used by operations in these two interfaces, ■ abstract data types passed as parameters to and returned as values from operations in these two interfaces, and ■ the interface for the XFN standard syntax model for parsing compound names. |
| FILES | <code>/usr/include/xfn/xfn.h</code> |
| SEE ALSO | <code>FN_ctx_t(3N)</code> , <code>FN_status_t(3N)</code> , <code>xfn_attributes(3N)</code> , <code>xfn_composite_names(3N)</code> , <code>xfn_compound_names(3N)</code> , <code>xfn_status_codes(3N)</code> , <code>fns(5)</code> , <code>fns_policies(5)</code> |
| NOTES | <p>The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.</p> |

| | |
|--------------------|---|
| NAME | xfn_attributes – an overview of XFN attribute operations |
| DESCRIPTION | <p>XFN assumes the following model for attributes. A set of zero or more attributes is associated with a named object. Each attribute in the set has a unique attribute identifier, an attribute syntax, and a (possibly empty) set of distinct data values. Each attribute value has an opaque data type. The attribute identifier serves as a name for the attribute. The attribute syntax indicates how the value is encoded in the buffer.</p> <p>The operations of the base attribute interface may be used to examine and modify the settings of attributes associated with existing named objects. These objects may be contexts or other types of objects. The attribute operations do not create names or remove names from contexts.</p> <p>The range of support for attribute operations may vary widely. Some naming systems may not support any attribute operations. Other naming systems may only support read operations, or operations on attributes whose identifiers are in some fixed set. A naming system may limit attributes to have a single value, or may require at least one value. Some naming systems may only associate attributes with context objects, while others may allow associating attributes with non-context objects.</p> <p>These are the interfaces:</p> <pre> #include <xfn/xfn.h> FN_attribute_t *fn_attr_get(FN_ctx_t *ctx, const FN_composite_name_t *name, const FN_identifier_t *attribute_id, FN_status_t *status); int fn_attr_modify(FN_ctx_t *ctx, const FN_composite_name_t *name, unsigned int mod_op, const FN_attribute_t *attr, FN_status_t *status); FN_attrset_t *fn_attr_get_ids(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status); FN_valuelist_t *fn_attr_get_values(FN_ctx_t *ctx, const FN_composite_name_t *name, const FN_identifier_t *attribute_id, FN_status_t *status); FN_attrvalue_t *fn_valuelist_next(FN_valuelist_t *vl, FN_identifier_t **attr_syntax, FN_status_t *status); void fn_valuelist_destroy(FN_valuelist_t *vl, FN_status_t *status); FN_multigetlist_t *fn_attr_multi_get(FN_ctx_t *ctx, const FN_composite_name_t *name, const FN_attrset_t *attr_ids, FN_status_t *status); </pre> |
| | (continued) |

(Continuation)

```

FN_attribute_t *fn_multigetlist_next(FN_multigetlist_t *ml,
    FN_status_t *status);

void fn_multigetlist_destroy(FN_multigetlist_t *ml, FN_status_t *status);

int fn_attr_multi_modify(FN_ctx_t *ctx, const FN_composite_name_t *name,
    const FN_attrmodlist_t *mods, FN_status_t *status,
    FN_attrmodlist_t **unexecuted_mods);

FN_attrset_t *fn_ctx_get_syntax_attrs(FN_ctx_t *ctx,
    const FN_composite_name_t *name, FN_status_t *status);

```

The following describes briefly the operations in the base attribute interface. Detailed descriptions are given in the respective reference manual pages for these operations.

fn_attr_get() returns the attribute identified. **fn_attr_modify()** modifies the attribute identified as described by *mod_op*.

fn_attr_get_ids() returns the identifiers of the attributes of the named object.

fn_attr_get_values() and its set of related operations are used for returning the individual values of an attribute.

fn_attr_multi_get() and its set of related operations are used for returning the requested attributes associated with the named object. **fn_attr_multi_modify()** modifies multiple attributes associated with the named object in a single invocation.

fn_ctx_get_syntax_attrs() returns the syntax attributes associated with the named context.

ERRORS

status is set as described in **FN_status_t(3N)** and **xfn_status_codes(3N)**. The following status codes are of special relevance to attribute operations:

| | |
|--------------------------|---|
| FN_E_ATTR_VALUE_REQUIRED | The operation attempted to create an attribute without a value, and the specific naming system does not allow this. |
| FN_E_ATTR_NO_PERMISSION | The caller did not have permission to perform the attempted attribute operation. |

| | |
|------------------------------|--|
| FN_E_INSUFFICIENT_RESOURCES | There are insufficient resources to retrieve the requested attribute(s). |
| FN_E_INVALID_ATTR_IDENTIFIER | The attribute identifier was not in a format acceptable to the naming system, or its contents was not valid for the format specified for the identifier. |
| FN_E_INVALID_ATTR_VALUE | One of the values supplied was not in the appropriate form for the given attribute. |
| FN_E_NO_SUCH_ATTRIBUTE | The object did not have an attribute with the given identifier. |
| FN_E_TOO_MANY_ATTR_VALUES | The operation attempted to associate more values with an attribute than the naming system supported. |

USAGE

Except for **fn_ctx_get_syntax_attrs()**, an attribute operation using a composite name is not necessarily equivalent to an independent **fn_ctx_lookup()** operation followed by an attribute operation in which the caller supplies the resulting reference and an empty name. This is because there is a range of attribute models in which an attribute is associated with a name in a context, or an attribute is associated with the object named, or both. XFN accommodates all of these alternatives. Invoking an attribute operation using the target context and the terminal atomic name accesses either the attributes that are associated with the target name or target named object; this is dependent on the underlying attribute model. This document uses the term *attributes associated with a named object* to refer to all of these cases.

XFN specifies no guarantees about the relationship between the attributes and the reference associated with a given name. Some naming systems may store the reference bound to a name in one or more attributes associated with a name. Attribute operations might affect the information used to construct a reference.

To avoid undefined results, programmers must use the operations in the context interface and not attribute operations when the intention is to manipulate a reference. Programmers should avoid the use of specific knowledge about how an XFN context implementation over a particular naming system constructs references.

SEE ALSO

`FN_attribute_t(3N)`, `FN_attrset_t(3N)`, `FN_attrvalue_t(3N)`,
`FN_composite_name_t(3N)`, `FN_ctx_t(3N)`, `FN_identifier_t(3N)`,
`FN_status_t(3N)`, `fn_attr_get(3N)`, `fn_attr_get_ids(3N)`,
`fn_attr_get_values(3N)`, `fn_attr_modify(3N)`,
`fn_attr_multi_get(3N)`, `fn_attr_multi_modify(3N)`,
`fn_ctx_get_syntax_attrs(3N)`, `fn_ctx_lookup(3N)`, `xfn(3N)`,
`xfn_status_codes(3N)`

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

| | |
|--------------------------|---|
| NAME | xfn_composite_names – XFN composite syntax: an overview of the syntax for XFN composite name |
| DESCRIPTION | <p>An <i>XFN composite name</i> consists of an ordered list of zero or more components. Each component is a string name from the namespace of a single naming system. It may be an atomic or a compound name in that namespace.</p> <p>XFN defines an abstract data type, <code>FN_composite_name_t</code>, for representing the structural form of a composite name. XFN also defines a standard string form for composite names. This form is the concatenation of the components of a composite name from left to right with the <i>XFN component separator</i> ('/') character to separate each component.</p> <p>These are the interfaces:</p> <pre>#include <xfn/xfn.h> FN_composite_name_t *fn_composite_name_from_string(const FN_string_t *str); FN_string_t *fn_string_from_composite_name(const FN_composite_name_t *name);</pre> <p>The function <code>fn_composite_name_from_string</code> parses the string representation of a composite name into its corresponding composite name object <code>FN_composite_name_t</code>. The function <code>fn_string_from_composite_name</code> composes the string representation of a composite name given its composite name object form <code>FN_composite_name_t</code>.</p> |
| APPLICATION USAGE | <p>Special characters used in the XFN composite name syntax, such as the separator or escape characters, have the same encoding as they would in ISO 646.</p> <p>All XFN implementations are required to support the portable representation, ISO 646. All other representations are optional.</p> <p>All characters of the string form of a XFN composite name use a single encoding. This does not preclude component names of a composite name in its structural form from having different encodings. Code set mismatches that occur during the process of converting a composite name structure to its string form are resolved in an implementation-dependent way. When an implementation discovers that a composite name has components with incompatible code sets, it returns the error code <code>FN_E_INCOMPATIBLE_CODE_SETS</code>.</p> |
| SEE ALSO | <code>FN_string_t</code> (3N), <code>FN_compound_name_t</code> (3N), <code>xfn</code> (3N) |

| | | |
|--------------------|---|--|
| NAME | xfn_compound_names – XFN compound syntax: an overview of XFN model for compound name parsing | |
| DESCRIPTION | <p>Each naming system in an XFN federation has a naming convention. XFN defines a standard model of expressing compound name syntax that covers a large number of specific name syntaxes and is expressed in terms of syntax properties of the naming convention.</p> <p>The model uses the attributes in the following table to describe properties of the syntax. Unless otherwise qualified, these syntax attributes have attribute identifiers that use the FN_ID_STRING format. A context that supports the XFN standard syntax model has an attribute set containing the <code>fn_syntax_type</code> (with identifier format FN_ID_STRING) attribute with the value "standard" (ASCII attribute syntax).</p> <p>These are the interfaces:</p> <pre data-bbox="495 882 1469 1029">#include <xfn/xfn.h> FN_attrset_t *fn_ctx_get_syntax_attrs(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status); FN_compound_name_t *fn_compound_name_from_syntax_attrs(const FN_attrset_t *aset, const FN_string_t *name, FN_status_t *status);</pre> | |
| | <code>fn_syntax_type</code> | <p>Its value is the ASCII string "standard" if the context supports the XFN standard syntax model. Its value is an implementation-specific value if another syntax model is supported.</p> |
| | <code>fn_std_syntax_direction</code> | <p>Its value is an ASCII string, one of "left_to_right", "right_to_left", or "flat". This determines whether the order of components in a compound name string goes from left to right, right to left, or whether the namespace is flat (in other words, not hierarchical; em all names are atomic).</p> |
| | <code>fn_std_syntax_separator</code> | <p>Its value is the separator string for this name syntax. This attribute is required unless the</p> |

| | |
|--|---|
| | <code>fn_std_syntax_direction</code> is "flat". |
| <code>fn_std_syntax_escape</code> | If present, its value is the escape string for this name syntax. |
| <code>fn_std_syntax_case_insensitive</code> | If this attribute is present, it indicates that names that differ only in case are considered identical. If this attribute is absent, it indicates that case is significant. If a value is present, it is ignored. |
| <code>fn_std_syntax_begin_quote</code> | If present, its value is the begin-quote string for this syntax. There can be multiple values for this attribute. |
| <code>fn_std_syntax_end_quote</code> | If present, its value is the end-quote string for this syntax. There can be multiple values for this attribute. |
| <code>fn_std_syntax_ava_separator</code> | If present, its value is the attribute value assertion separator string for this syntax. |
| <code>fn_std_syntax_typeval_separator</code> | If present, its value is the attribute type-value separator string for this syntax. |
| <code>fn_std_syntax_code_sets</code> | If present, its value identifies the code sets of the string representation for this syntax. Its value consists of a structure containing an array of code sets supported by the context; the first member of the array is the preferred code set of the context. The values for the code sets are defined in the X/Open code set registry. If this attribute is not present, or if the value is empty, the default code set is |

| | |
|--|---|
| <code>fn_std_syntax_locale_info</code> | ISO 646 (same encoding as ASCII). If present, identifies locale information, such as character set information, of the string representation for this syntax. The interpretation of its value is implementation-dependent. |
|--|---|

The XFN standard syntax attributes are interpreted according to the following rules:

1. In a string without quotes or escapes, any instance of the separator string delimits two atomic names.
2. A separator, quotation or escape string is escaped if preceded immediately (on the left) by the escape string.
3. A non-escaped begin-quote which precedes a component must be matched by a non-escaped end-quote at the end of the component. Quotes embedded in non-quoted names are treated as simple characters and do not need to be matched. An unmatched quotation fails with the status code `FN_E_ILLEGAL_NAME`.
4. If there are multiple values for begin-quote and end-quote, a specific begin-quote value must be matched with its corresponding end-quote value.
5. When the separator appears between a (non-escaped) begin quote and the end quote, it is ignored.
6. When the separator is escaped, it is ignored. An escaped begin-quote or end-quote string is not treated as a quotation mark. An escaped escape string is not treated as an escape string.
7. A non-escaped escape string appearing within quotes is interpreted as an escape string. This can be used to embed an end-quote within a quoted string.

After constructing a compound name from a string, the resulting component atoms have one level of escape strings and quotations interpreted and consumed.

`fn_ctx_get_syntax_attrs()` is used to obtain the syntax attributes associated with a context.

`fn_compound_name_from_syntax()` is used to construct a compound name object using the string form of the name and the syntax attributes of the name.

ERRORS

| | |
|-----------------------------|--|
| FN_E_ILLEGAL_NAME | The name supplied to the operation was not a well-formed component according to the name syntax of the context. |
| FN_E_INCOMPATIBLE_CODE_SETS | Code set mismatches that occur during the construction of the compound name's string form are resolved in an implementation-dependent way. When an implementation discovers that a compound name has components with incompatible code sets, it returns this error code. |
| FN_E_INVALID_SYNTAX_ATTRS | The syntax attributes supplied are invalid or insufficient to fully specify the syntax. |
| FN_E_SYNTAX_NOT_SUPPORTED | The syntax specified is not supported. |

USAGE

Most applications treat names as opaque data. Hence, the majority of clients of the XFN interface will not need to parse compound names from specific naming systems. Some applications, however, such as browsers, need such capabilities. These applications would use `fn_ctx_get_syntax_attrs()` to obtain the syntax-related attributes of a context and, if the context uses the XFN standard syntax model, it would examine these attributes to determine the name syntax of the context.

SEE ALSO

`FN_attribute_t(3N)`, `FN_attrset_t(3N)`, `FN_compound_name_t(3N)`, `FN_identifier_t(3N)`, `FN_string_t(3N)` `fn_ctx_get_syntax_attrs(3N)`, `xfn(3N)`

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

| | |
|--------------------|---|
| NAME | xfn_links - XFN links: an overview of XFN links |
| DESCRIPTION | <p>An <i>XFN link</i> is a special form of reference that contains a composite name, the <i>link name</i>, and that may be bound to an atomic name in an XFN context. Because the link name is a composite name, it may span multiple namespaces.</p> <p>Normal resolution of names in context operations always follows XFN links. If the first composite name component of the link name is the atomic name ".", the link name is resolved relative to the same context in which the link is bound, otherwise, the link name is resolved relative to the XFN Initial Context of the client. The link name may itself cause resolution to pass through other XFN links. This gives rise to the possibility of a cycle of links whose resolution could not terminate normally. As a simple means to avoid such non-terminating resolutions, implementations may define limits on the number of XFN links that may be resolved in any single operation invoked by the caller.</p> <p>These are the interfaces:</p> <pre> #include <xfn/xfn.h> FN_ref_t *fn_ref_create_link(const FN_composite_name_t *link_name); int fn_ref_is_link(const FN_ref_t *ref); FN_composite_name_t *fn_ref_link_name(const FN_ref_t *link_ref); FN_ref_t *fn_ctx_lookup_link(FN_ctx_t *ctx, const FN_composite_name_t *name, FN_status_t *status); unsigned int fn_status_link_code(const FN_status_t *stat); const FN_composite_name_t *fn_status_link_remaining_name(const FN_status_t *stat); const FN_composite_name_t *fn_status_link_resolved_name(const FN_status_t *stat); const FN_ref_t *fn_status_link_resolved_ref(const FN_status_t *stat); int fn_status_set_link_code(FN_status_t *stat, unsigned int code); int fn_status_set_link_remaining_name(FN_status_t *stat, const FN_composite_name_t *name); int fn_status_set_link_resolved_name(FN_status_t *stat, const FN_composite_name_t *name); int fn_status_set_link_resolved_ref(FN_status_t *stat, const FN_ref_t *ref); </pre> |

Links are bound to names using the normal **fn_ctx_bind()** and unbound using the normal **fn_ctx_unbind()** operation. The operation **fn_ref_create_link()** is provided for constructing a link reference from a composite name. Since normal resolution always follows links, a separate operation, **fn_ctx_lookup_link()** is provided to lookup the link itself.

In the case that an error occurred while resolving an XFN link, the status object set by the operation contains additional information about that error and sets the corresponding link status fields using **fn_status_set_link_code()**, **fn_status_set_link_remaining_name()**, **fn_status_set_link_resolved_name()** and **fn_status_set_link_resolved_ref()**. The link status fields can be retrieved using **fn_status_link_code()**, **fn_status_link_remaining_name()**, **fn_status_link_resolved_name()** and **fn_status_link_resolved_ref()**.

ERRORS

The following status codes are of special relevance when performing operations involving XFN links:

| | |
|-----------------------------|--|
| FN_E_LINK_ERROR | There was an error encountered resolving an XFN link encountered during resolution of the supplied name. Check the link part of the status object to determine cause of the link error. |
| FN_E_LINK_LOOP_LIMIT | A non-terminating loop (cycle) in the resolution can arise due to XFN links encountered during the resolution of a composite name. This code indicates either the definite detection of such a cycle, or that resolution exceeded an implementation-defined limit on the number of XFN links allowed for a single operation invoked by the caller. |
| FN_E_MALFORMED_LINK | A malformed link reference was encountered. For the fn_ctx_lookup_link() operation, the name supplied resolved to a reference that was not a link. |

APPLICATION USAGE

For the **fn_ctx_bind()**, **fn_ctx_unbind()**, **fn_ctx_rename()**, **fn_ctx_lookup_link()**, **fn_ctx_create_subcontext()** and **fn_ctx_destroy_subcontext()** operations, resolution of the given name continues to the target context — that named by all but the terminal atomic part of the given name; the terminal atomic name is not resolved. Consequently, for operations that involve unbinding the terminal atomic part such as **fn_ctx_unbind()**, if the terminal atomic name is bound to a link, the link is not followed and the link itself is unbound from the terminal atomic name.

Many naming systems support a native notion of link that may be used within the naming system itself. XFN does not determine whether there is any relationship between such native links and XFN links.

SEE ALSO

`FN_composite_name_t(3N)`, `FN_ref_t(3N)`, `FN_status_t(3N)`,
`fn_ctx_bind(3N)`, `fn_ctx_destroy_subcontext(3N)`,
`fn_ctx_lookup(3N)`, `fn_ctx_lookup_link(3N)`, `fn_ctx_rename(3N)`,
`fn_ctx_unbind(3N)`, `xfn_status_codes(3N)`, `xfn(3N)`

| | | | | | | | | | | | | | | | |
|--|--|-------------------------|--------------------------|--------------------------------------|--|---------------------------------------|---|--|---|---|--|---------------------------------------|---|----------------------------|---|
| NAME | xfn_status_codes – descriptions of XFN status codes | | | | | | | | | | | | | | |
| SYNOPSIS | <pre>#include <xfn/xfn.h></pre> | | | | | | | | | | | | | | |
| DESCRIPTION | <p>The result status of operations in the context interface and the attribute interface is encapsulated in an <code>FN_status_t</code> object. This object contains information about how the operation completed: whether an error occurred in performing the operation; if so, what kind of error; and information localizing where the error occurred. In the case that the error occurred while resolving an XFN link, the status object contains additional information about that error.</p> <p>The context status object consists of several items of information. One of them is the primary status code, describing the disposition of the operation. In the case that an error occurred while resolving an XFN link, the primary status code has the value <code>FN_E_LINK_ERROR</code>, and the link status code describes the error that occurred while resolving the XFN link.</p> | | | | | | | | | | | | | | |
| XFN Status Codes | <p>Both the primary status code and the link status code are values of type <code>unsigned int</code> that are drawn from the same set of meaningful values. XFN reserves the values 0 through 127 for standard meanings. Currently, values and interpretations for the following codes are determined by XFN.</p> <table border="0"> <tr> <td style="padding-right: 20px;"><code>FN_SUCCESS</code></td> <td>The operation succeeded.</td> </tr> <tr> <td style="padding-right: 20px;"><code>FN_E_ATTR_NO_PERMISSION</code></td> <td>The caller did not have permission to perform the attempted attribute operation.</td> </tr> <tr> <td style="padding-right: 20px;"><code>FN_E_ATTR_VALUE_REQUIRED</code></td> <td>The operation attempted to create an attribute without a value, and the specific naming system does not allow this.</td> </tr> <tr> <td style="padding-right: 20px;"><code>FN_E_AUTHENTICATION_FAILURE</code></td> <td>The identity of the client principal could not be verified.</td> </tr> <tr> <td style="padding-right: 20px;"><code>FN_E_COMMUNICATION_FAILURE</code></td> <td>An error occurred in communicating with one of the contexts involved in the operation.</td> </tr> <tr> <td style="padding-right: 20px;"><code>FN_E_CONFIGURATION_ERROR</code></td> <td>A problem was detected that indicated an error in the installation of the XFN implementation.</td> </tr> <tr> <td style="padding-right: 20px;"><code>FN_E_CONTINUE</code></td> <td>The operation should be continued using the remaining name and the resolved reference returned in the status.</td> </tr> </table> | <code>FN_SUCCESS</code> | The operation succeeded. | <code>FN_E_ATTR_NO_PERMISSION</code> | The caller did not have permission to perform the attempted attribute operation. | <code>FN_E_ATTR_VALUE_REQUIRED</code> | The operation attempted to create an attribute without a value, and the specific naming system does not allow this. | <code>FN_E_AUTHENTICATION_FAILURE</code> | The identity of the client principal could not be verified. | <code>FN_E_COMMUNICATION_FAILURE</code> | An error occurred in communicating with one of the contexts involved in the operation. | <code>FN_E_CONFIGURATION_ERROR</code> | A problem was detected that indicated an error in the installation of the XFN implementation. | <code>FN_E_CONTINUE</code> | The operation should be continued using the remaining name and the resolved reference returned in the status. |
| <code>FN_SUCCESS</code> | The operation succeeded. | | | | | | | | | | | | | | |
| <code>FN_E_ATTR_NO_PERMISSION</code> | The caller did not have permission to perform the attempted attribute operation. | | | | | | | | | | | | | | |
| <code>FN_E_ATTR_VALUE_REQUIRED</code> | The operation attempted to create an attribute without a value, and the specific naming system does not allow this. | | | | | | | | | | | | | | |
| <code>FN_E_AUTHENTICATION_FAILURE</code> | The identity of the client principal could not be verified. | | | | | | | | | | | | | | |
| <code>FN_E_COMMUNICATION_FAILURE</code> | An error occurred in communicating with one of the contexts involved in the operation. | | | | | | | | | | | | | | |
| <code>FN_E_CONFIGURATION_ERROR</code> | A problem was detected that indicated an error in the installation of the XFN implementation. | | | | | | | | | | | | | | |
| <code>FN_E_CONTINUE</code> | The operation should be continued using the remaining name and the resolved reference returned in the status. | | | | | | | | | | | | | | |

| | |
|-------------------------------|---|
| FN_E_CTX_NO_PERMISSION | The client did not have permission to perform the operation. |
| FN_E_CTX_NOT_EMPTY | (Applies only to fn_ctx_destroy_subcontext() .) The naming system required that the context be empty before its destruction, and it was not empty. |
| FN_E_CTX_UNAVAILABLE | Service could not be obtained from one of the contexts involved in the operation. This may be because the naming system is busy, or is not providing service. In some implementations this may not be distinguished from a communication failure. |
| FN_E_ILLEGAL_NAME | The name supplied to the operation was not a well- formed XFN composite name, or one of the component names was not well-formed according to the syntax of the naming system(s) involved in its resolution. |
| FN_E_E_INCOMPATIBLE_CODE_SETS | The operation involved character strings of incompatible code sets, or the supplied code set is not supported by the implementation. |
| FN_E_INSUFFICIENT_RESOURCES | Either the client or one of the involved contexts could not obtain sufficient resources (for example, memory, file descriptors, communication ports, stable media space, and so on) to complete the operation successfully. |
| FN_E_INVALID_ATTR_IDENTIFIER | The attribute identifier was not in a format acceptable to the naming system, or its content was not valid for the format specified for the identifier. |

| | |
|---------------------------|--|
| FN_E_INVALID_ATTR_VALUE | One of the values supplied was not in the appropriate form for the given attribute. |
| FN_E_INVALID_ENUM_HANDLE | The enumeration handle supplied was invalid, either because it was from another enumeration, or because an update operation occurred during the enumeration, or because of some other reason. |
| FN_E_INVALID_SYNTAX_ATTRS | The syntax attributes supplied are invalid or insufficient to fully specify the syntax. |
| FN_E_LINK_ERROR | There was an error in resolving an XFN link encountered during resolution of the supplied name. |
| FN_E_LINK_LOOP_LIMIT | A non-terminating loop (cycle) in the resolution can arise due to XFN links encountered during the resolution of a composite name. This code indicates either the definite detection of such a cycle, or that resolution exceeded an implementation-defined limit on the number of XFN links allowed for a single operation invoked by the caller. |
| FN_E_MALFORMED_LINK | A malformed link reference was encountered. For fn_ctx_lookup_link() , the name supplied resolved to a reference that was not a link. |
| FN_E_MALFORMED_REFERENCE | A context object could not be constructed from the supplied reference, because the reference was not properly formed. |
| FN_E_NAME_IN_USE | (Only for operations that bind names.) The supplied name was already in use. |

| | |
|------------------------------|--|
| FN_E_NAME_NOT_FOUND | Resolution of the supplied composite name proceeded to a context in which the next atomic component of the name was not bound. |
| FN_E_NO_SUCH_ATTRIBUTE | The object did not have an attribute with the given identifier. |
| FN_E_NO_SUPPORTED_ADDRESS | A context object could not be constructed from a particular reference. The reference contained no address type over which the context interface was supported. |
| FN_E_NOT_A_CONTEXT | Either one of the intermediate atomic names did not name a context, and resolution could not proceed beyond this point, or the operation required that the caller supply the name of a context, and the name did not resolve to a reference for a context. |
| FN_E_OPERATION_NOT_SUPPORTED | The operation attempted is not supported. |
| FN_E_PARTIAL_RESULT | The operation attempted is returning a partial result. |
| FN_E_SYNTAX_NOT_SUPPORTED | The syntax type specified is not supported. |
| FN_E_TOO_MANY_ATTR_VALUES | The operation attempted to associate more values with an attribute than the naming system supported. |
| FN_E_UNSPECIFIED_ERROR | An error occurred that could not be classified by any of the other error codes. |

FILES

#include <xfn/xfn.h> XFN status codes header file

SEE ALSO

FN_status_t(3N), **xfn(3N)**

NOTES

The implementation of XFN in this Solaris release is based on the X/Open preliminary specification. It is likely that there will be minor changes to these interfaces to reflect changes in the final version of this specification. The next minor release of Solaris will offer binary compatibility for applications developed using the current interfaces. As the interfaces evolve toward standardization, it is possible that future releases of Solaris will require minor source code changes to applications that have been developed against the preliminary specification.

| | |
|----------------------|---|
| NAME | y0, y1, yn – Bessel functions of the second kind |
| SYNOPSIS | <pre>cc [flag ...] file ... -lm [library ...] double y0(double x); double y1(double x); double yn(int n, double x);</pre> |
| DESCRIPTION | The y0() , y1() and yn() functions compute Bessel functions of <i>x</i> of the second kind of orders 0, 1 and <i>n</i> respectively. The value of <i>x</i> must be positive. |
| RETURN VALUES | <p>Upon successful completion, y0() , y1() and yn() will return the relevant Bessel value of <i>x</i> of the second kind.</p> <p>If <i>x</i> is NaN, NaN is returned.</p> <p>If the <i>x</i> argument to y0() , y1() or yn() is negative, <code>-HUGE_VAL</code> or NaN is returned, and <code>errno</code> may be set to <code>EDOM</code> .</p> <p>If <i>x</i> is 0.0, <code>-HUGE_VAL</code> is returned and <code>errno</code> may be set to <code>ERANGE</code> or <code>EDOM</code> .</p> <p>If the correct result would cause overflow, <code>-HUGE_VAL</code> is returned and <code>errno</code> may be set to <code>ERANGE</code> .</p> <p>For exceptional cases, matherr(3M) tabulates the values to be returned as dictated by Standards other than XPG4.</p> |
| ERRORS | <p>The y0() , y1() and yn() functions may fail if:</p> <p>EDOM The value of <i>x</i> is negative.</p> <p>ERANGE The value of <i>x</i> is too large in magnitude, or <i>x</i> is 0.0, or the correct result would cause overflow.</p> |
| USAGE | An application wishing to check for error situations should set <code>errno</code> to 0 before calling y0() , y1() or yn() . If <code>errno</code> is non-zero on return, or the return value is NaN, an error has occurred. |

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | MT-Safe |

SEE ALSO

`isnan(3M)` , `j0(3M)` , `matherr(3M)` , `attributes(5)` , `standards(5)`

| | |
|--------------------|--|
| NAME | ypclnt, yp_get_default_domain, yp_bind, yp_unbind, yp_match, yp_first, yp_next, yp_all, yp_order, yp_master, yperr_string, ypprot_err - NIS Version 2 client interface |
| SYNOPSIS | <code>cc [<i>flag</i> ...] <i>file</i> ... -lnsl [<i>library</i> ...] #include <rpcsvc/ypclnt.h> #include <rpcsvc/yp_prot.h></code> |
| DESCRIPTION | <p>This package of functions provides an interface to NIS, Network Information Service Version 2, formerly referred to as YP. In this version of SunOS, NIS version 2 is supported only for compatibility with previous versions. The recommended enterprise level information service is NIS+ or NIS version 3, see <code>nis+(1)</code>. Moreover, this version of SunOS supports only the client interface to NIS version 2. It is expected that this client interface will be served either by an existing <code>ypserv</code> process running on another machine on the network that has an earlier version of SunOS or by an NIS+ server, see <code>rpc.nisd(1M)</code>, running in "YP-compatibility mode". Refer to the NOTES section in <code>ypfiles(4)</code> for implications of being an NIS client of an NIS+ server in "YP-compatibility mode", and to <code>ypbind(1M)</code>, <code>ypwhich(1)</code>, <code>ypmatch(1)</code>, and <code>ypcat(1)</code> for commands to access NIS from a client machine. The package can be loaded from the standard library, <code>/usr/lib/libnsl.so.1</code>.</p> <p>All input parameter names begin with <i>in</i>. Output parameters begin with <i>out</i>. Output parameters of type <code>char **</code> should be addresses of uninitialized character pointers. Memory is allocated by the NIS client package using <code>malloc(3C)</code>, and may be freed by the user code if it has no continuing need for it. For each <i>outkey</i> and <i>outval</i>, two extra bytes of memory are allocated at the end that contain NEWLINE and null, respectively, but these two bytes are not reflected in <i>outkeylen</i> or <i>outvallen</i>. <i>indomain</i> and <i>inmap</i> strings must be non-null and null-terminated. String parameters which are accompanied by a count parameter may not be null, but may point to null strings, with the count parameter indicating this. Counted strings need not be null-terminated.</p> <p>All functions in this package of type <i>int</i> return 0 if they succeed, and a failure code (<code>YPERR_ xxxx</code>) otherwise. Failure codes are described in the ERRORS section.</p> |
| Routines | <p>yp_bind (char * <i>indomain</i>);</p> <p>To use the NIS name services, the client process must be "bound" to an NIS server that serves the appropriate domain using <code>yp_bind()</code>. Binding need not be done explicitly by user code; this is done automatically whenever an NIS lookup function is called. <code>yp_bind()</code> can be called directly for processes that make use of a backup strategy (for example, a local file) in cases when NIS services are not available. If a process calls <code>yp_bind()</code>, it should call <code>yp_unbind()</code> when it is done using NIS in order to free up resources.</p> <p>void yp_unbind(char * <i>indomain</i>);</p> |

Each binding allocates (uses up) one client process socket descriptor; each bound domain costs one socket descriptor. However, multiple requests to the same domain use that same descriptor. **yp_unbind()** is available at the client interface for processes that explicitly manage their socket descriptors while accessing multiple domains. The call to **yp_unbind()** makes the domain *unbound*, and frees all per-process and per-node resources used to bind it.

If an RPC failure results upon use of a binding, that domain will be unbound automatically. At that point, the **ypclnt()** layer will retry a few more times or until the operation succeeds, provided that **rpcbind(1M)** and **ypbind(1M)** are running, and either

- the client process cannot bind a server for the proper domain, or
- RPC requests to the server fail.

If an error is not RPC-related, or if **rpcbind** is not running, or if **ypbind** is not running, or if a bound **ypserv** process returns any answer (success or failure), the **ypclnt** layer will return control to the user code, either with an error code, or a success code and any results.

yp_get_default_domain (char ** *outdomain*);

The NIS lookup calls require a map name and a domain name, at minimum. It is assumed that the client process knows the name of the map of interest. Client processes should fetch the node's default domain by calling **yp_get_default_domain()**, and use the returned *outdomain* as the *indomain* parameter to successive NIS name service calls. The domain thus returned is the same as that returned using the **SI_SRPC_DOMAIN** command to the **sysinfo(2)** system call. The value returned in *outdomain* should not be freed.

yp_match(char * *indomain* , char * *inmap* , char * *inkey* , int *inkeylen* , char ** *outval* , int * *outvallen*);

yp_match() returns the value associated with a passed key. This key must be exact; no pattern matching is available. **yp_match()** requires a full YP map name; for example, *hosts.byname* instead of the nickname *hosts*.

yp_first(char * *indomain* , char * *inmap* , char ** *outkey* , int * *outkeylen* , char ** *outval* , int * *outvallen*);

yp_first() returns the first key-value pair from the named map in the named domain.

yp_next(char * *indomain* , char * *inmap* , char * *inkey* , int *inkeylen* , char ** *outkey* , int * *outkeylen* , char ** *outval* , int * *outvallen*);

yp_next() returns the next key-value pair in a named map. The *inkey* parameter must be the *outkey* returned from an initial call to **yp_first()** (to get the second key-value pair) or the one returned from the *n*th call to **yp_next()** (to get the *n*th + second key-value pair). Similarly, the *inkeylen* parameter must be the *outkeylen* returned from the earlier **yp_first()** or **yp_next()** call.

The concept of first (and, for that matter, of next) is particular to the structure of the NIS map being processing; there is no relation in retrieval order to either the lexical order within any original (non-NIS name service) data base, or to any obvious numerical sorting order on the keys, values, or key-value pairs. The only ordering guarantee made is that if the **yp_first()** function is called on a particular map, and then the **yp_next()** function is repeatedly called on the same map at the same server until the call fails with a reason of `YPERR_NOMORE`, every entry in the data base will be seen exactly once. Further, if the same sequence of operations is performed on the same map at the same server, the entries will be seen in the same order.

Under conditions of heavy server load or server failure, it is possible for the domain to become unbound, then bound once again (perhaps to a different server) while a client is running. This can cause a break in one of the enumeration rules; specific entries may be seen twice by the client, or not at all. This approach protects the client from error messages that would otherwise be returned in the midst of the enumeration. The next paragraph describes a better solution to enumerating all entries in a map.

yp_all(char * *indomain* , char * *inmap* , struct ypall_callback * *incallback*);

The function **yp_all()** provides a way to transfer an entire map from server to client in a single request using TCP (rather than UDP as with other functions in this package). The entire transaction take place as a single RPC request and response. **yp_all()** can be used just like any other NIS name service procedure, identify the map in the normal manner, and supply the name of a function which will be called to process each key-value pair within the map. The call to **yp_all()** returns only when the transaction is completed (successfully or unsuccessfully), or the **foreach()** function decides that it does not want to see any more key-value pairs.

The third parameter to **yp_all()** is

```
struct ypall_callback *incallback {
    \011int (*foreach)();
    \011char *data;
};
```

The function **foreach()** is called

```

foreach(int
  instatus
  , char *
  inkey
  ,
  int
  inkeylen
  , char *
  inval
  ,
  int
  invallen
  , char *
  indata
  );

```

The *instatus* parameter will hold one of the return status values defined in `<rpcsvc/yp_prot.h>` — either `YP_TRUE` or an error code. (See **ypprot_err()**, below, for a function which converts an NIS name service protocol error code to a `ypclnt` layer error code.)

The key and value parameters are somewhat different than defined in the synopsis section above. First, the memory pointed to by the *inkey* and *inval* parameters is private to the **yp_all()** function, and is overwritten with the arrival of each new key-value pair. It is the responsibility of the **foreach()** function to do something useful with the contents of that memory, but it does not own the memory itself. Key and value objects presented to the **foreach()** function look exactly as they do in the server's map — if they were not NEWLINE-terminated or null-terminated in the map, they will not be here either.

The *indata* parameter is the contents of the *incallback* ⇒ *data* element passed to **yp_all()**. The *data* element of the callback structure may be used to share state information between the **foreach()** function and the mainline code. Its use is optional, and no part of the NIS client package inspects its contents — cast it to something useful, or ignore it.

The **foreach()** function is a Boolean. It should return 0 to indicate that it wants to be called again for further received key-value pairs, or non-zero to stop the flow of key-value pairs. If **foreach()** returns a non-zero value, it is not called again; the functional value of **yp_all()** is then 0.

yp_order(char * *indomain* , char * *inmap* , unsigned long * *outorder*);

yp_order() returns the order number for a map. This function is not supported if the `ypbind` process on the client's system is bound to an NIS+ server running in "YP-compatibility mode".

yp_master(char * *indomain* , char * *inmap* , char ** *outname*);

yp_master() returns the machine name of the master NIS server for a map.

char *yperr_string(int *incode*);

yperr_string() returns a pointer to an error message string that is null-terminated but contains no period or NEWLINE.

ypprot_err (unsigned int *incode*);

ypprot_err() takes an NIS name service protocol error code as input, and returns a ypclnt layer error code, which may be used in turn as an input to **yperr_string()** .

RETURN VALUES

All integer functions return 0 if the requested operation is successful, or one of the following errors if the operation fails.

| | |
|---------------|--|
| YPERR_ACCESS | Access violation. |
| YPERR_BADARGS | The arguments to the function are bad. |
| YPERR_BADDB | The YP database is bad. |
| YPERR_BUSY | The database is busy. |
| YPERR_DOMAIN | Cannot bind to server on this domain. |
| YPERR_KEY | No such key in map. |
| YPERR_MAP | No such map in server's domain. |
| YPERR_NODOM | Local domain name not set. |
| YPERR_NOMORE | No more records in map database. |
| YPERR_PMAP | Cannot communicate with <code>rpcbind</code> . |
| YPERR_RESRC | Resource allocation failure. |
| YPERR_RPC | RPC failure; domain has been unbound. |
| YPERR_YPBIND | Cannot communicate with <code>ypbind</code> . |
| YPERR_YPERR | Internal YP server or client error. |
| YPERR_YPSESV | Cannot communicate with <code>ypserv</code> . |
| YPERR_VERS | YP version mismatch. |

FILES

/usr/lib/libnsl.so.1

ATTRIBUTES

See **attributes(5)** for descriptions of the following attributes:

| ATTRIBUTE TYPE | ATTRIBUTE VALUE |
|----------------|-----------------|
| MT-Level | Safe |

SEE ALSO

nis+(1), **ypcat(1)**, **ypmatch(1)**, **ypwhich(1)**, **rpc.nisd(1M)**, **rpcbind(1M)**, **ypbind(1M)**, **ypserv(1M)**, **sysinfo(2)**, **malloc(3C)**, **ypfiles(4)**, **attributes(5)**

| NAME | yp_update – change NIS information | | | | |
|----------------------|---|----------------|-----------------|----------|--------|
| SYNOPSIS | #include <rpcsvc/ypclnt.h> int yp_update (char *domain, char *map, unsigned ypop, char *key, int keylen, char *data, int datalen); | | | | |
| DESCRIPTION | yp_update() is used to make changes to the NIS database. The syntax is the same as that of yp_match() except for the extra parameter <i>ypop</i> which may take on one of four values. If it is POP_CHANGE then the data associated with the key will be changed to the new value. If the key is not found in the database, then yp_update() will return YPERR_KEY. If <i>ypop</i> has the value YPOP_INSERT then the key-value pair will be inserted into the database. The error YPERR_KEY is returned if the key already exists in the database. To store an item into the database without concern for whether it exists already or not, pass <i>ypop</i> as YPOP_STORE and no error will be returned if the key already or does not exist. To delete an entry, the value of <i>ypop</i> should be YPOP_DELETE. This routine depends upon secure RPC, and will not work unless the network is running secure RPC. | | | | |
| RETURN VALUES | If the value of <i>ypop</i> is POP_CHANGE, yp_update() returns the error YPERR_KEY if the key is not found in the database. If the value of <i>ypop</i> is POP_INSERT, yp_update() returns the error YPERR_KEY if the key already exists in the database. | | | | |
| ATTRIBUTES | See attributes(5) for descriptions of the following attributes: | | | | |
| | <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th> <th>ATTRIBUTE VALUE</th> </tr> </thead> <tbody> <tr> <td>MT-Level</td> <td>Unsafe</td> </tr> </tbody> </table> | ATTRIBUTE TYPE | ATTRIBUTE VALUE | MT-Level | Unsafe |
| ATTRIBUTE TYPE | ATTRIBUTE VALUE | | | | |
| MT-Level | Unsafe | | | | |
| SEE ALSO | secure_rpc(3N) , ypclnt(3N) , attributes(5) | | | | |
| NOTES | This interface is unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread. | | | | |

Index

A

- abandon an LDAP operation in progress —
 - ldap_abandon, 1238
- abort — terminate the process abnormally, 165
- abs — return absolute value of integer, 166
- absolute value function — fabs, 713
- accept — accept a connection on a socket, 167
- access dynamic thread scheduling
 - thr_getprio, 2259
 - thr_setprio, 2259
- access dynamic thread scheduling parameters
 - pthread_getschedparam, 1717
 - pthread_setschedparam, 1717
- access utmpx file entry
 - endutxent, 1096
 - getutmp, 1096
 - getutmpx, 1096
 - getutxent, 1096
 - getutxid, 1096
 - getutxline, 1096
 - pututxline, 1096
 - setutxent, 1096
 - updwtmp, 1096
 - updwtmpx, 1096
 - utmpxname, 1096
- accounting
 - time accounting for current process —
 - times, 2289
- acos — arc cosine function, 179
- acosh — inverse hyperbolic functions, 180
- acquire and release stream lock —
 - flockfile, 740
 - funlockfile, 740
- activate audio-visual alarm
 - beep, 240
 - flash, 240
- add a wchar_t character (with attributes) to a curses window and advance cursor — curs_addwch, 368, 371, 373
 - addnwstr, 373
 - addwch, 368
 - addwchnstr, 371
 - addwchstr, 371
 - addwstr, 373
 - echowchar, 368
 - mvaddnwstr, 373
 - mvaddwch, 368
 - mvaddwchnstr, 371
 - mvaddwchstr, 371
 - mvaddwstr, 373
 - mvwaddnwstr, 373
 - mvwaddwch, 368
 - mvwaddwchnstr, 371
 - mvwaddwchstr, 371
 - mvwaddwstr, 373
 - waddnwstr, 373
 - waddwch, 368
 - waddwchnstr, 371
 - waddwchstr, 371
 - waddwstr, 373
 - wechowchar, 368
- add a character (with rendition) to a window
 - addch, 182
 - mvaddch, 182
 - mvwaddch, 182

- waddch, 182
- add a complex character (with rendition) to a window
 - add_wch, 193
 - mvadd_wch, 193
 - mvwadd_wch, 193
 - wadd_wch, 193
- add a complex character and refresh window
 - echo_wchar, 636
 - wecho_wchar, 636
- add a multi-byte character string (without rendition) to a window
 - addnstr, 186
 - addstr, 186
 - mvaddnstr, 186
 - mvaddstr, 186
 - mvwaddstr, 186
 - mwwaddnstr, 186
 - waddnstr, 186
 - waddstr, 186
- add a single-byte border to a window
 - border, 261
 - box, 261
 - wborder, 261
- add a single-byte character and refresh window
 - echochar, 635
 - wechochar, 635
- add a wide-character string to a window
 - addnwstr, 188
 - addwstr, 188
 - mvaddnwstr, 188
 - mvaddwstr, 188
 - mvwaddnwstr, 188
 - mvwaddwstr, 188
 - waddnwstr, 188
 - waddwstr, 188
- add character and refresh window
 - pechochar, 1631
 - pecho_wchar, 1631
- addch — add a character (with rendition) to a window, 182
- addchnstr — copy a character string (with renditions) to a window, 184
- addchstr — copy a character string (with renditions) to a window, 184
- additional severities
 - define — addsev, 190
- addnstr — add a multi-byte character string (without rendition) to a window, 186
- addnwstr — add a wide-character string to a window, 188, 373
- address in an XFN reference
 - fn_ref_addr_assign, 835
 - fn_ref_addr_copy, 835
 - fn_ref_addr_create, 835
 - fn_ref_addr_data, 835
 - fn_ref_addr_description, 835
 - fn_ref_addr_destroy, 835
 - fn_ref_addr_length, 835
 - FN_ref_addr_t, 835
 - fn_ref_addr_type, 835
- address of symbol
 - get address in shared object — dlsym, 582
- addsev — define additional severities, 190
- addseverity — build a list of severity levels for an application for use with fmtmsg, 191
- addstr — add a multi-byte character string (without rendition) to a window, 186
- addwch — add a wchar_t character (with attributes) to a curses window and advance cursor, 368
- addwchnstr — add string of wchar_t characters (and attributes) to a curses window, 371
- addwchstr — add string of wchar_t characters (and attributes) to a curses window, 371
- addwstr — add a wide-character string to a window, 188, 373
- add_wch — add a complex character (with rendition) to a window, 193
- add_wchnstr — copy a string of complex characters (with renditions) to a window, 195
- add_wchstr — copy a string of complex characters (with renditions) to a window, 195
- adjcurspos — moving the cursor by character, 374
- advance — regular expression compile and match routines, 1825

- aiocancel — cancel an asynchronous operation, 197
- aioread — read or write asynchronous I/O operations, 204
- aiowait — wait for completion of asynchronous I/O operation, 214
- aiowrite — read or write asynchronous I/O operations, 204
- aio_cancel — cancel asynchronous I/O request, 198
- aio_fsync — asynchronous file synchronization, 202
- aio_read — asynchronous read and write operations, 207
- aio_return — retrieve return status of asynchronous I/O operation, 210
- aio_suspend — wait for asynchronous I/O request, 212
- aio_write — asynchronous write to a file, 216
- alarm
 - schedule signal after interval in microseconds — ualarm, 2437
- ALE curses library, *see* curses library,
- allocate and deallocate process handles for libthread_db
 - td_ta_delete, 2198
 - td_ta_get_ph, 2198
 - td_ta_new, 2198
- allocate or deallocate a buffer for trace data
 - tnftcl_buffer_alloc, 2299
 - tnftcl_buffer_dealloc, 2299
- alphasort — scan a directory, 1935
- annotate source code with info for tools
 - NOTE, 1564
 - _NOTE, 1564
- applications
 - build a list of severity levels for use with fmtmsg — addseverity, 191
 - display a message on stderr or system console — fmtmsg, 745
 - get entries from symbol table — nlist, 1554
- apply padding information and output string
 - putp, 1774
 - tputs, 1774
- arc — graphics interface, 1638
- arc cosine function — acos, 179
- arc sine function — asin, 219
- arc tangent function — atan2, 221, 223
- arithmetic
 - compute the quotient and remainder — div, 559
- arithmetic, 48-bit integer
 - generate uniformly distributed pseudo-random numbers — drand48, 629
- asctime — convert date and time to string, 2103
- asin — arc sine function, 219
- asinh — inverse hyperbolic functions, 180
- assert — verify program assertion, 220
- associate a stream with a file descriptor — fdopen, 723
- asynchronous file synchronization
 - aio_sync, 202
- asynchronous I/O
 - aiocancel, 197
 - aiowait, 214
 - aio_cancel, 198
 - retrieve return status — aio_return, 210
- asynchronous read and write operations
 - aio_read, aio_write, 207
- asynchronous write to a file — aio_write, 216
- asysmem — return physical memory information, 2147
- atan — arc tangent function, 223
- atan2 — arc tangent function, 221
- atanh — inverse hyperbolic functions, 180
- atexit — add program termination routine, 224
- atof — convert string to double-precision number, 2120
- atoi — string conversion routines, 2122
- atol — string conversion routines, 2122
- atoll — string conversion routines, 2122
- attribute modifications, list of
 - fn_attrmodlist_add, 775
 - fn_attrmodlist_assign, 775
 - fn_attrmodlist_copy, 775
 - fn_attrmodlist_count, 775
 - fn_attrmodlist_create, 775
 - fn_attrmodlist_destroy, 775
 - fn_attrmodlist_first, 775
 - fn_attrmodlist_next, 775
 - FN_attrmodlist_t, 775

- attribute search options
 - `fn_search_control_assign`, 842
 - `fn_search_control_copy`, 842
 - `fn_search_control_create`, 842
 - `fn_search_control_destroy`, 842
 - `fn_search_control_follow_links`, 842
 - `fn_search_control_max_names`, 842
 - `fn_search_control_return_attr_ids`, 842
 - `fn_search_control_return_ref`, 842
 - `fn_search_control_scope`, 842
 - `FN_search_control_t`, 842
- `attdoff` — change foreground window attributes, 227, 376
- `attdon` — change foreground window attributes, 227, 376
- `attdset` — change foreground window attributes, 227, 376
- `attd_get` — control window attributes, 225
- `attd_off` — control window attributes, 225
- `attd_on` — control window attributes, 225
- `attd_set` — control window attributes, 225
- audit control file information
 - `endac`, 960
 - `getacdir`, 960
 - `getacflg`, 960
 - `getacinfo`, 960
 - `getacmin`, 960
 - `getacna`, 960
 - `setac`, 960
- audit record tokens, creating
 - `au_to_attr`, 234
 - `au_to_data`, 234
 - `au_to_groups`, 234
 - `au_to_in_addr`, 234
 - `au_to_in_ipc`, 234
 - `au_to_in_ipc_perm`, 234
 - `au_to_iport`, 234
 - `au_to_me`, 234
 - `au_to_opaque`, 234
 - `au_to_path`, 234
 - `au_to_process`, 234
 - `au_to_return`, 234
 - `au_to_socket`, 234
 - `au_to_subject`, 234
 - `au_to_text`, 234
- audit record tokens, manipulating
 - `au_close`, 229
 - `au_open`, 229
 - `au_preselect`, 231
 - `au_write`, 229
- authentication information routines for PAM
 - `pam_get_item`, 1598
 - `pam_set_item`, 1598
- authentication transaction routines for PAM
 - `pam_end`, 1615
 - `pam_start`, 1615
- `authnone_create` — library routines for client side remote procedure call authentication, 1856
- `authsys_create` — library routines for client side remote procedure call authentication, 1856
- `authsys_create_default` — library routines for client side remote procedure call authentication, 1856
- `auth_destroy` — library routines for client side remote procedure call authentication, 1856
- `au_close` — construct audit records, 229
- `au_open` — construct audit records, 229
- `au_preselect` — preselect an audit record, 231
- `au_to_arg` — creating audit record tokens, 234
- `au_to_attr` — creating audit record tokens, 234
- `au_to_data` — creating audit record tokens, 234
- `au_to_groups` — creating audit record tokens, 234
- `au_to_in_addr` — creating audit record tokens, 234
- `au_to_ipc` — creating audit record tokens, 234
- `au_to_ipc_perm` — creating audit record tokens, 234
- `au_to_iport` — creating audit record tokens, 234
- `au_to_me` — creating audit record tokens, 234
- `au_to_opaque` — creating audit record tokens, 234
- `au_to_path` — creating audit record tokens, 234
- `au_to_process` — creating audit record tokens, 234
- `au_to_return` — creating audit record tokens, 234
- `au_to_text` — creating audit record tokens, 234
- `au_user_mask` — get user's binary preselection mask, 236

au_write — write audit records, 229

B

base 10 logarithm function — log10, 1345

base-64 ASCII characters

convert from long integer — l64a, 164

basename — return the last element of path name, 238

Basic Encoding Rules library decoding functions

- ber_alloc_t, 242
- ber_bvdup, 242
- ber_bvecfree, 242
- ber_bvfree, 242
- ber_decode, 242
- ber_first_element, 242
- ber_flatten, 242
- ber_free, 242
- ber_get_bitstring, 242
- ber_get_boolean, 242
- ber_get_int, 242
- ber_get_next, 242
- ber_get_null, 242
- ber_get_stringa, 242
- ber_get_stringal, 242
- ber_get_stringb, 242
- ber_init, 242
- ber_next_element, 242
- ber_peek_tag, 242
- ber_scanf, 242
- ber_skiptag, 242

Basic Security Module functions

- au_close, 229
- au_open, 229
- au_preselect, 231
- au_to_attr, 234
- au_to_data, 234
- au_to_groups, 234
- au_to_in_addr, 234
- au_to_ipc, 234
- au_to_ipc_perm, 234
- au_to_iport, 234
- au_to_me, 234
- au_to_opaque, 234
- au_to_path, 234
- au_to_process, 234
- au_to_return, 234

- au_to_socket, 234
- au_to_subject, 234
- au_to_text, 234
- au_user_mask, 236
- au_write, 229

baudrate — return terminal baud rate, 239

bcmp — operates on variable length strings of bytes, 272

bcopy — operates on variable length strings of bytes, 272

beep — activate audio-visual alarm, 240

ber_alloc — simplified Basic Encoding Rules library encoding functions, 247

ber_alloc_t — Basic Encoding Rules library decoding functions, 242

ber_bvdup — Basic Encoding Rules library decoding functions, 242

ber_bvecfree — Basic Encoding Rules library decoding functions, 242

ber_bvfree — Basic Encoding Rules library decoding functions, 242

ber_decode — Basic Encoding Rules library decoding functions, 242

ber_encode — simplified Basic Encoding Rules library encoding functions, 247

ber_first_element — Basic Encoding Rules library decoding functions, 242

ber_flatten — Basic Encoding Rules library decoding functions, 242

ber_flush — simplified Basic Encoding Rules library encoding functions, 247

ber_free — Basic Encoding Rules library decoding functions, 242

ber_get_bitstring — Basic Encoding Rules library decoding functions, 242

ber_get_boolean — Basic Encoding Rules library decoding functions, 242

ber_get_int — Basic Encoding Rules library decoding functions, 242

ber_get_next — Basic Encoding Rules library decoding functions, 242

ber_get_null — Basic Encoding Rules library decoding functions, 242
 ber_get_stringa — Basic Encoding Rules library decoding functions, 242
 ber_get_stringal — Basic Encoding Rules library decoding functions, 242
 ber_get_stringb — Basic Encoding Rules library decoding functions, 242
 ber_init — Basic Encoding Rules library decoding functions, 242
 ber_next_element — Basic Encoding Rules library decoding functions, 242
 ber_peek_tag — Basic Encoding Rules library decoding functions, 242
 ber_printf — simplified Basic Encoding Rules library encoding functions, 247
 ber_put_bitstring — simplified Basic Encoding Rules library encoding functions, 247
 ber_put_boolean — simplified Basic Encoding Rules library encoding functions, 247
 ber_put_int — simplified Basic Encoding Rules library encoding functions, 247
 ber_put_null — simplified Basic Encoding Rules library encoding functions, 247
 ber_put_ostring — simplified Basic Encoding Rules library encoding functions, 247
 ber_put_seq — simplified Basic Encoding Rules library encoding functions, 247
 ber_put_set — simplified Basic Encoding Rules library encoding functions, 247
 ber_put_string — simplified Basic Encoding Rules library encoding functions, 247
 ber_scanf — Basic Encoding Rules library decoding functions, 242
 ber_skiptag — Basic Encoding Rules library decoding functions, 242
 ber_start_seq — simplified Basic Encoding Rules library encoding functions, 247
 ber_start_set — simplified Basic Encoding Rules library encoding functions, 247
 Bessel functions of the first kind
 — j0, 1184
 — j1, 1184
 — jn, 1184
 Bessel functions of the second kind
 — y0, 2553
 — y1, 2553
 — yn, 2553
 bgets — read stream up to next delimiter, 251
 binary search of sorted table
 — bsearch, 270
 binary search trees, manage
 — tdelete, 2398
 — tfind, 2398
 — tsearch, 2398
 — twalk, 2398
 bind — bind a name to a socket, 252
 bind a reference to a name — fn_ctx_bind, 804
 bind a reference to a name and associate attributes with named object
 — fn_attr_bind, 752
 bind or unbind the current thread with the door server pool
 — door_bind, 612
 — door_unbind, 612
 bindtextdomain — select location of domain, 1082
 bit and byte operations
 find first set bit — ffs, 728
 bkgd — set the background character (and rendition) of window, 257
 bkgdset — set the background character (and rendition) of window, 257
 bkgrnd — set or get the background character (and rendition) of window using a complex character, 259
 bkgrndset — set or get the background character (and rendition) of

- chgat, 294
- mvchgat, 294
- mvwchgat, 294
- wchgat, 294
- character based forms package
 - forms, 902
- character based menus package
 - menus, 1427
- character based panels package
 - panels, 1622
- character handling
 - ctype, 357
 - isalnum, 357
 - isalpha, 357
 - isascii, 357
 - iscntrl, 357
 - isdigit, 357
 - isgraph, 357
 - islower, 357
 - isprint, 357
 - ispunct, 357
 - isspace, 357
 - isupper, 357
 - isxdigit, 357
- character string
 - fn_string_assign, 859
 - fn_string_bytecount, 859
 - fn_string_charcount, 859
 - fn_string_code_set, 859
 - fn_string_compare, 859
 - fn_string_compare_substring, 859
 - fn_string_contents, 859
 - fn_string_copy, 859
 - fn_string_create, 859
 - fn_string_destroy, 859
 - fn_string_from_contents, 859
 - fn_string_from_str, 859
 - fn_string_from_strings, 859
 - fn_string_from_str_n, 859
 - fn_string_from_substring, 859
 - fn_string_is_empty, 859
 - fn_string_next_substring, 859
 - fn_string_prev_substring, 859
 - fn_string_str, 859
 - FN_string_t, 859
- check for type-ahead characters —
 - typeahead, 2436
- check whether or not Volume Management is
 - managing a pathname —
 - volmgt_inuse, 2455
- check whether specific Volume Management
 - features are enabled —
 - volmgt_feature_enabled, 2454
- chgat — change the rendition of characters in
 - a window, 294
- Cipher Block Chaining
 - fast CBC encryption — cbc_crypt, 520
- circle — graphics interface, 1638
- class-dependent data translation
 - elf32_xlatetof, 648
 - elf32_xlatetom, 648
 - elf64_xlatetof, 648
 - elf64_xlatetom, 648
- cldap_close — dispose of connectionless
 - LDAP pointer, 296
- cldap_open — LDAP connectionless
 - communication
 - preparation, 297
- cldap_search_s — connectionless LDAP
 - search, 298
 - Retransmission Algorithm, 298
- cldap_setretryinfo — set connectionless LDAP
 - request retransmission
 - parameters, 300
- clear — clear a window, 301
- clear a window
 - clear, 301
 - erase, 301
 - wclear, 301
 - werase, 301
- clear to the end of a line
 - clrtoeol, 309
 - wclrtoeol, 309
- clear to the end of a window
 - clrtobot, 308
 - wclrtobot, 308
- clearok — set terminal output controls, 302
- client side remote procedure call
 - authentication, library
 - routines for
 - authnone_create, 1856
 - authsys_create, 1856
 - authsys_create_default, 1856
 - auth_destroy, 1856

- `rpc_clnt_auth`, 1856
- `clnt_call` — library routines for client side calls, 1858
- `clnt_control` — library routines for dealing with creation and manipulation of CLIENT handles, 1862
- `clnt_create` — library routines for dealing with creation and manipulation of CLIENT handles, 1862
- `clnt_create_timed` — library routines for dealing with creation and manipulation of CLIENT handles, 1862
- `clnt_create_vers` — library routines for dealing with creation and manipulation of CLIENT handles, 1862
- `clnt_create_vers_timed` — library routines for dealing with creation and manipulation of CLIENT handles, 1862
- `clnt_destroy` — library routines for dealing with creation and manipulation of CLIENT handles, 1862
- `clnt_dg_create` — library routines for dealing with creation and manipulation of CLIENT handles, 1862
- `clnt_freeres` — library routines for client side calls, 1858
- `clnt_geterr` — library routines for client side calls, 1858
- `clnt_pcreateerror` — library routines for dealing with creation and manipulation of CLIENT handles, 1862
- `clnt_perrno` — library routines for client side calls, 1858
- `clnt_perror` — library routines for client side calls, 1858
- `clnt_raw_create` — library routines for dealing with creation and manipulation of CLIENT handles, 1862
- `clnt_spcreateerror` — library routines for dealing with creation and manipulation of CLIENT handles, 1862
- `clnt_sperrno` — library routines for client side calls, 1858
- `clnt_sperror` — library routines for client side calls, 1858
- `clnt_tli_create` — library routines for dealing with creation and manipulation of CLIENT handles, 1862
- `clnt_tp_create` — library routines for dealing with creation and manipulation of CLIENT handles, 1862
- `clnt_tp_create_timed` — library routines for dealing with creation and manipulation of CLIENT handles, 1862
- `clnt_vc_create` — library routines for dealing with creation and manipulation of CLIENT handles, 1862
- `clock` — report CPU time used, 304
- `clock_getres` — high-resolution clock operations, 305
- `clock_gettime` — high-resolution clock operations, 305
- `clock_settime` — high-resolution clock operations, 305
- close a directory stream — `closedir`, 307
- close a shared object — `dlclose`, 566
- close a stream — `fclose`, 718
- close a `tnftcl` handle — `tnftcl_close`, 2302
- `closedir` — close a directory stream, 307
- `closelog` — control system log, 2142
- `closepl` — graphics interface, 1638
- `closevt` — graphics interface, 1638
- `clrtoebot` — clear to the end of a window, 308
- `clrtoeol` — clear to the end of a line, 309
- code conversion allocation function — `iconv_open`, 1136
- code conversion deallocation function — `iconv_close`, 1135
- code conversion for Process Code and File Code
 - `strtows`, 2127
 - `wstostr`, 2127

code conversion function — iconv, 1130

collect target process statistics for libthread_db

- td_ta_enable_stats, 2187
- td_ta_get_stats, 2187
- td_ta_reset_stats, 2187

color_content — manipulate color information, 283

COLOR_PAIR — manipulate color information, 283

color_set — control window attributes, 225

column positions of a wide-character code — wcwidth, 2510

column positions of a wide-character string — wcswidth, 2503

command options

- get option letter from argument vector — getopt, 1030

command suboptions

- parse suboptions from a string — getsubopt, 1079

commands

- open, close to and from a command — p2open, p2close, 1574
- return stream to remote — rcmd, 1787

communications

- accept a connection on a socket — accept, 167
- allocate memory for, 2153
- bind a name to a socket — bind, 252
- create a pair of connected sockets — socketpair, 2070
- create an endpoint for communication — socket, 2064
- get name of peer connected to socket — getpeername, 1036
- get socket name — getsockname, 1065
- initiate a connection on a socket — connect, 332
- listen for connections on a socket — listen, 1333
- scatter data in order to test the network — spray, 2074
- send a message from a socket — send, sendto, sendmsg, 1989
- shut down part of a full-duplex connection — shutdown, 2028

compare thread IDs — pthread_equal, 1712

compare wide-characters in memory — wmemcmp, 2512

compile — regular expression compile and match routines, 1825

compile and execute regular expressions

- re_comp, 1799
- re_exec, 1799

component names spanning multiple naming systems

- fn_composite_name_append_comp, 794
- fn_composite_name_append_name, 794
- fn_composite_name_assign, 794
- fn_composite_name_copy, 794
- fn_composite_name_count, 794
- fn_composite_name_create, 794
- fn_composite_name_delete_comp, 794
- fn_composite_name_destroy, 794
- fn_composite_name_first, 794
- fn_composite_name_from_str, 794
- fn_composite_name_from_string, 794
- fn_composite_name_insert_comp, 794
- fn_composite_name_insert_name, 794
- fn_composite_name_is_empty, 794
- fn_composite_name_is_equal, 794
- fn_composite_name_is_prefix, 794
- fn_composite_name_is_suffix, 794
- fn_composite_name_last, 794
- fn_composite_name_next, 794
- fn_composite_name_prefix, 794
- fn_composite_name_prepend_comp, 794
- fn_composite_name_prepend_name, 794
- fn_composite_name_prev, 794
- fn_composite_name_suffix, 794
- FN_composite_name_t, 794
- fn_string_from_composite_name, 794

compute natural logarithm — log1p, 1346

computes exponential functions — expm1, 712

concepts related to condition variables — condition, 315

concepts relating to mutual exclusion locks — mutex, 1472

condition — concepts related to condition variables, 315

Condition Signaling, 316

- Condition Wait, 315
- Destroy, 316
- Initialize, 315
- condition variables
 - cond_broadcast, 310
 - cond_destroy, 310
 - cond_init, 310
 - cond_signal, 310
 - cond_timedwait, 310
 - cond_wait, 310
- cond_broadcast — condition variables, 310
- cond_destroy — condition variables, 310
- cond_init — condition variables, 310
 - Condition Signaling, 312
 - Condition Wait, 311
 - Destroy, 312
 - Initialize, 310
- cond_signal — condition variables, 310
- cond_timedwait — condition variables, 310
- cond_wait — condition variables, 310
- configuration administration interface
 - config_ap_id_cmp, 318
 - config_change_state, 318
 - config_list, 318
 - config_private_func, 318
 - config_stat, 318
 - config_strerror, 318
 - config_test, 318
 - config_unload, 318
- configuration script
 - execute — doconfig, 609
- config_ap_id_cmp — configuration administration interface, 318
- config_change_state — configuration administration interface, 318
- config_list — configuration administration interface, 318
- config_private_func — configuration administration interface, 318
- config_stat — configuration administration interface, 318
- config_strerror — configuration administration interface, 318
- config_test — configuration administration interface, 318
- config_unload — configuration administration interface, 318
- confstr — get configurable variables, 326
- connect — initiate a connection on socket, 332
- connect to a DMI service provider
 - ConnectToServer, 339, 558
- connectionless LDAP search —
 - cldap_search_s, 298
- construct a handle to a context object using the given reference —
 - fn_ctx_handle_from_ref, 817
- construct equivalent name in same context —
 - fn_ctx_equivalent_name, 809
- cont — graphics interface, 1638
- control flush of input and output on interrupt
 - noqiflush, 1563
 - qiflush, 1563
- control kernel tracing and process filtering
 - tnftcl_filter_list_add, 2330
 - tnftcl_filter_list_delete, 2330
 - tnftcl_filter_list_get, 2330
 - tnftcl_filter_state_set, 2330
 - tnftcl_trace_state_set, 2330
- control probes of another process where caller provides /proc functionality
 - tnftcl_check_libs, 2304
 - tnftcl_indirect_open, 2304
- control system log
 - closelog, 2142
 - openlog, 2142
 - setlogmask, 2142
 - syslog, 2142
- control window attributes
 - attr_get, 225
 - attr_off, 225
 - attr_on, 225
 - attr_set, 225
 - color_set, 225
 - wattr_get, 225
 - wattr_off, 225
 - wattr_on, 225
 - wattr_set, 225
 - wcolor_set, 225
- control window refresh
 - is_linetouched, 1175
 - is_wintouched, 1175
 - touchline, 1175
 - touchwin, 1175
 - untouchwin, 1175
 - wtouchln, 1175

- convert wide character string to
 - double-precision number —
wcstod, 2490 to 2125
 - ascftime, 2103
 - cftime, 2103
 - watof, 2490
 - wstod, 2490
- convert a character string to a wide-character string — mbstowcs, 1384
- convert a character string to a wide-character string (restartable) — mbsrtowcs, 1382
- convert a character to a wide-character code — mbtowc, 1385
- convert a character to a wide-character code (restartable) — mbrtowc, 1379
- convert a supplied name into an absolute pathname that can be used to access removable media — media_findname, 1389
- convert a thread id or thread address to a thread handle
 - td_ta_map_addr2thr, 2196
 - td_ta_map_id2thr, 2196
- convert a wide character to printable form — wunctrl, 2523
- convert a wide-character code to a character — wctomb, 2507
- convert a wide-character code to a character (restartable) — wctomb, 2483
- convert a wide-character string to a character string — wcstombs, 2495
- convert a wide-character string to a character string (restartable) — wcsrtombs, 2487
- convert between Volume Management symbolic names, and the devices that correspond to them
 - volmgt_symdev, 2462
 - volmgt_symname, 2462
- convert character to printable form — unctrl, 2438
- convert date and time to wide character string — wcsftime, 2486
- convert floating-point number to string
 - ecvt, 639
 - fcvt, 639
 - gcvt, 639
- convert formatted input
 - fscanf, 1937
 - scanf, 1937
 - sscanf, 1937
- convert formatted wide-character input
 - fwscanf, 947
 - swscanf, 947
 - wscanf, 947
- convert monetary value to string —strfmon, 2098
- convert numbers to strings
 - econvert, 637
 - ecvt, 637
 - fconvert, 637
 - fcvt, 637
 - fprintf, 1647
 - gconvert, 637
 - gcvt, 637
 - printf, 1647
 - qconvert, 637
 - qfconvert, 637
 - qgconvert, 637
 - seconvert, 637
 - sfconvert, 637
 - sgconvert, 637
 - sprintf, 1647
 - vfprintf, 1647
 - vprintf, 1647
- convert string to double-precision number
 - atof, 2120
 - strtod, 2120
- convert to wchar_t strings
 - wsprintf, 2520
- convert wide character string to unsigned long — wcstoul, 2496
- copy a character string (with renditions) to a window
 - addchnstr, 184
 - addchstr, 184
 - mvaddchnstr, 184
 - mvaddchstr, 184
 - mvwaddchnstr, 184
 - mvwaddchstr, 184
 - waddchnstr, 184
 - waddchstr, 184

- copy a string of complex characters (with renditions) to a window
 - add_wchnstr, 195
 - add_wchstr, 195
 - mvadd_wchnstr, 195
 - mvadd_wchstr, 195
 - mvwadd_wchnstr, 195
 - mvwadd_wchstr, 195
 - wadd_wchnstr, 195
 - wadd_wchstr, 195
- copy wide-characters in memory — wmemcpy, 2513
- copy wide-characters in memory with overlapping areas — wmemmove, 2514
- copysign — return magnitude of first argument and sign of second argument, 341
- copywin — overlay or overwrite any portion of window, 342
- cos — cosine function, 344
- cosh — hyperbolic cosine function, 345
- cosine function — cos, 344
- cplus_demangle — decode a C++ encoded symbol name, 516
- CPU time
 - report for calling process — clock, 304
- CPU-use
 - prepare execution profile — monitor, 1447
- create a door descriptor — door_create, 618
- create a GSS_API security context
 - rpc_gss_seccreate, 1882
- create a new window or subwindow
 - derwin, 518
 - newwin, 518
 - subwin, 518
- create a temporary file — tmpfile, 2296
- create a thread — pthread_create, 1707
- create a tread — thr_create, 2240
- create cancellation point in the calling thread.
 - pthread_testcancel, 1769
- create handle for internal process probe control
 - tnftcl_internal_open, 2307
- create handle for kernel probe control — tnftcl_kernel_open, 2309
- create new file from dynamic object component — dldump, 567
- create or refresh a pad or subpad
 - newpad, 1504
 - pnoutrefresh, 1504
 - prefresh, 1504
 - subpad, 1504
- create subcontext and associate attributes — fn_attr_create_subcontext, 754
- create thread-specific data key — pthread_key_create, 1723
- CRT handling and optimization package
 - curses, 391
- crypt — string encoding function, 346, 347
- cset — get information on EUC codesets, 349
- csetcol — get information on EUC codesets, 349
- csetlen — get information on EUC codesets, 349
- csetno — get information on EUC codesets, 349
- ctermid — generate path name for controlling terminal, 351
- ctermid_r — generate path name for controlling terminal, 351
- ctype — character handling, 357
- cube root function — cbrt, 290
- current location of a named directory stream — telldir, 2220
- current working directory
 - get pathname — getcwd, 980
- curses — CRT handling and optimization package, 391, 407, 468
 - Attributes, Color Pairs, and Renditions, 410
 - Complex Characters, 412
 - Data Types, 408
 - Display Operations, 413
 - Input Processing, 417
 - newpad, 468
 - Non-Spacing Characters, 412
 - Overlapping Windows, 413
 - pechochar, 468
 - pechowchar, 468
 - pnoutrefresh, 468
 - preresh, 468
 - Screens, Windows, and Terminals, 409
 - Special Characters, 416
 - subpad, 468
- curses bell and screen flash routines

- beep, 378
- curs_beep, 378
- flash, 378
- curses borders, horizontal and vertical lines,
 - create
 - border, 381
 - box, 381
 - curs_border, 381
 - wborder, 381
 - whline, 381
 - wvline, 381
- curses character and window attribute control
 - routines
 - attroff, 376
 - attron, 376
 - attrset, 376
 - curs_attr, 376
 - standend, 376
 - standout, 376
 - wattroff, 376
 - wattron, 376
 - wattrset, 376
 - wstandend, 376
 - wstandout, 376
- curses color manipulation routines
 - can_change_colors, 385
 - color_content, 385
 - curs_color, 385
 - has_colors, 385
 - init_color, 385
 - init_pair, 385
 - pair_content, 385
 - start_color, 385
- curses cursor and window coordinates
 - curs_getyx, 433
 - getbegyx, 433
 - getmaxyx, 433
 - getparyx, 433
 - getyx, 433
- curses environment query routines
 - baudrate, 482
 - curs_termattrs, 482
 - erasechar, 482
 - has_ic, 482
 - has_il, 482
 - killchar, 482
 - longname, 482
 - termattrs, 482
 - termname, 482
- curses interfaces to termcap library
 - curs_termcap, 484
 - tgetent, 484
 - tgetflag, 484
 - tgetnum, 484
 - tgetstr, 484
 - tgoto, 484
 - tputs, 484
- curses interfaces to terminfo database
 - curs_terminfo, 486
 - del_curterm, 486
 - mvcur, 486
 - putp, 486
 - restartterm, 486
 - setterm, 486
 - setupterm, 486
 - set_curterm, 486
 - tigetflag, 486
 - tigetnum, 486
 - tigetstr, 486
 - tparm, 486
 - tputs, 486
 - vidattr, 486
 - vidputs, 486
- curses library
 - See also* form library, menu library, or panel library,
 - adjcurspos, 374
 - curs_alecompat, 374
 - movenextch, 374
 - moveprevch, 374
 - wadjcurspos, 374
 - wmovenextch, 374
 - wmoveprevch, 374
- curses miscellaneous utility routines
 - curs_util, 492
 - delay_output, 492
 - filter, 492
 - flushinp, 492
 - getwin, 492
 - keyname, 492
 - putwin, 492
 - unctrl, 492
 - use_env, 492
- curses refresh control routines
 - curs_touch, 490

- is_linetouched, 490
- is_wintouched, 490
- touchline, 490
- touchwin, 490
- untouchwin, 490
- wtouchln, 490
- curses screen initialization and manipulation
 - routines
 - curs_initscr, 439
 - delscreen, 439
 - endwin, 439
 - initscr, 439
 - isendwin, 439
 - newterm, 439
 - set_term, 439
- curses screen, read/write from/to file
 - curs_scr_dump, 476
 - scr_dump, 476
 - scr_init, 476
 - scr_restore, 476
 - scr_set, 476
- curses soft label routines
 - curs_slk, 480
 - slk_attroff, 480
 - slk_attron, 480
 - slk_attrset, 480
 - slk_clear, 480
 - slk_init, 480
 - slk_label, 480
 - slk_noutrefresh, 480
 - slk_refresh, 480
 - slk_restore, 480
 - slk_set, 480
 - slk_touch, 480
- curses terminal input option control routines
 - cbreak, 441
 - curs_inopts, 441
 - echo, 441
 - halfdelay, 441
 - intrflush, 441
 - keypad, 441
 - meta, 441
 - nocbreak, 441
 - nodelay, 441
 - noecho, 441
 - noqiflush, 441
 - noraw, 441
 - notimeout, 441
 - qiflush, 441
 - raw, 441
 - timeout, 441
 - typeahead, 441
 - wtimeout, 441
- curses terminal keyboard
 - curs_getstr, 425
 - getstr, 425
 - mvgetstr, 425
 - mvwgetstr, 425
 - wgetnstr, 425
 - wgetstr, 425
- curses terminal keyboard, get characters
 - curs_getch, 420
 - getch, 420
 - mvgetch, 420
 - mvwgetch, 420
 - ungetch, 420
 - wgetch, 420
- curses terminal output option control routines
 - clearok, 463
 - curs_oupt, 463
 - idcok, 463
 - idlok, 463
 - immedok, 463
 - leaveok, 463
 - nl, 463
 - nonl, 463
 - scrollok, 463
 - setscreg, 463
 - wsetscreg, 463
- curses window background manipulation
 - routines
 - bkgd, 379
 - bkgdset, 379
 - curs_bkgd, 379
 - wbkgd, 379
 - wbkgdset, 379
- curses window cursor
 - curs_move, 462
 - move, 462
 - wmove, 462
- curses window, add character and advance
 - cursor
 - addch, 361
 - curs_addch, 361
 - echochar, 361

- mvwaddch, 361
- waddch, 361
- wechochar, 361
- curses window, add string of characters
 - addchnstr, 364
 - addchstr, 364
 - curs_addchstr, 364
 - mvaddchnstr, 364
 - mvaddchstr, 364
 - mvwaddchnstr, 364
 - mvwaddchstr, 364
 - waddchnstr, 364
 - waddchstr, 364
- curses window, add string of characters and advance cursor
 - addnstr, 366
 - addstr, 366
 - curs_addstr, 366
 - mvaddnstr, 366
 - mvaddstr, 366
 - mvwaddstr, 366
 - waddnstr, 366
 - waddstr, 366
- curses window, clear all or part
 - clear, 383
 - clrtoobot, 383
 - clrtoeol, 383
 - curs_clear, 383
 - erase, 383
 - wclear, 383
 - wclrtoobot, 383
 - wclrtoeol, 383
 - werase, 383
- curses window, convert formatted input
 - curs_scanw, 474
 - mvscanw, 474
 - mvwscanw, 474
 - scanw, 474
 - vwscanw, 474
 - wscanw, 474
- curses window, delete and insert lines
 - curs_deleteln, 389
 - deleteln, 389
 - insdelln, 389
 - insertln, 389
 - wdeleteln, 389
 - winsdelln, 389
- winsertln, 389
- curses window, delete character under cursor
 - curs_delch, 388
 - delch, 388
 - mvdelch, 388
 - mvwdelch, 388
 - wdelch, 388
- curses window, get character and its attributes
 - curs_inch, 435
 - inch, 435
 - mvinch, 435
 - mvwinch, 435
 - winch, 435
- curses window, get string of characters
 - curs_inchstr, 437
 - curs_instr, 448
 - inchnstr, 437
 - inchstr, 437
 - innstr, 448
 - instr, 448
 - mvinchnstr, 437
 - mvinchstr, 437
 - mvinnstr, 448
 - mvinstr, 448
 - mvwinchnstr, 437
 - mvwinchstr, 437
 - mvwinstr, 448
 - winchnstr, 437
 - winchstr, 437
 - winnstr, 448
 - winstr, 448
- curses window, insert character before character under cursor
 - curs_insch, 445
 - insch, 445
 - mvinsch, 445
 - mvwinsch, 445
 - winsch, 445
- curses window, insert string before character under cursor
 - curs_instr, 446
 - insnstr, 446
 - instr, 446
 - mvinsnstr, 446
 - mvinsnstr, 446
 - mvwinsnstr, 446

- mvwinsstr, 446
- winsnstr, 446
- winsstr, 446
- curses window, scroll
 - curs_scroll, 478
 - scl, 478
 - scroll, 478
 - wscrl, 478
- curses windows and lines, refresh
 - curs_refresh, 472
 - doupdate, 472
 - redrawwin, 472
 - refresh, 472
 - wnoutrefresh, 472
 - wredrawln, 472
 - wrefresh, 472
- curses windows, create
 - curs_window, 494
 - delwin, 494
 - derwin, 494
 - dupwin, 494
 - mvderwin, 494
 - mvwin, 494
 - newwin, 494
 - subwin, 494
 - syncok, 494
 - wcursyncup, 494
 - wsyncdown, 494
 - wsyncup, 494
- curses windows, overlap and manipulate
 - copywin, 466
 - curs_overlay, 466
 - overlay, 466
 - overwrite, 466
- curses windows, print formatted output
 - curs_printw, 470
 - mvprintw, 470
 - mvwprintw, 470
 - printw, 470
 - vwprintw, 470
 - wprintw, 470
- curses, low-level routines
 - curs_kernel, 459
 - curs_set, 459
 - def_prog_mode, 459
 - def_shell_mode, 459
 - getsyx, 459
 - napms, 459
 - resettty, 459
 - reset_prog_mode, 459
 - reset_shell_mode, 459
 - ripoffline, 459
 - savetty, 459
 - setsyx, 459
- curs_addwch — add a wchar_t character (with attributes) to a curses window and advance cursor, 368
- curs_addwchstr — add string of wchar_t characters (and attributes) to a curses window, 371
- curs_addwstr — add a string of wchar_t characters to a curses window and advance cursor, 373
- curs_alecompat — moving the cursor by character, 374
- curs_attr — curses character and window attribute control routines, 376
 - Attributes, 377
- curs_getwch — get (or push back) wchar_t characters from curses terminal keyboard, 426
 - Function Keys, 427
- curs_getwstr — get wchar_t character strings from curses terminal keyboard, 431
- curs_inswch — insert a wchar_t character before the character under the cursor in a curses window, 450
- curs_inswstr — insert wchar_t string before character under the cursor in a curses window, 451
- curs_inwch — get a wchar_t character and its attributes from a curses window, 453
- curs_inwchstr — get a string of wchar_t characters (and attributes) from a curses window, 455
- curs_inwstr — get a string of wchar_t characters from a curses window, 457
- curs_pad — create and display curses pads, 468
- curs_set — set visibility of cursor, 479

cuserid — get character-string representation
of login name of user, 497

D

data base subroutines — dbm, 499

- dbmclose, 499
- dbminit, 499
- delete, 499
- fetch, 499
- firstkey, 499
- nextkey, 499
- store, 499

Data Encryption Standard

- fast DES encryption — des_crypt, 520

database functions

- dbm_clearerr, 501
- dbm_close, 501
- dbm_delete, 501
- dbm_error, 501
- dbm_fetch, 501
- dbm_firstkey, 501
- dbm_nextkey, 501
- dbm_open, 501
- dbm_store, 501

date and time

- convert to string — asctime, 352
- convert user format date and time —
getdate, 982
- gettimeofday, 1088

date and time conversion — strptime, 2115

dbm — data base subroutines, 499

dbmclose — data base subroutines, 499

dbminit — data base subroutines, 499

dbm_clearerr — database functions, 501

dbm_close — database functions, 501

dbm_delete — database functions, 501

dbm_error — database functions, 501

dbm_fetch — database functions, 501

dbm_firstkey — database functions, 501

dbm_nextkey — database functions, 501

dbm_open — database functions, 501

dbm_store — database functions, 501

debugging memory allocator

- calloc, 2480
- cfree, 2480
- free, 2480
- mallinfo, 2480

- malloc, 2480
- mallopt, 2480
- memalign, 2480
- realloc, 2480
- valloc, 2480

decimal record from double-precision floating
— double_to_decimal, 736

decimal record to double-precision floating —
decimal_to_double, 506

decimal record to extended-precision floating
— decimal_to_extended, 506

decimal record to quadruple-precision floating
— decimal_to_quadruple, 506

decimal record to single-precision floating —
decimal_to_single, 506

decimal_to_double — decimal record to
double-precision floating, 506

decimal_to_extended — decimal record to
extended-precision
floating, 506

decimal_to_quadruple — decimal record to
quadruple-precision
floating, 506

decimal_to_single — decimal record to
single-precision floating, 506

decode a C++ encoded symbol name

- cplus_demangle, 516
- demangle, 516

decompose floating-point number

- modf, 1446
- modff, 1446

define character class — wctype, 2509

define character mapping — wctrans, 2508

define default catalog — setcat, 2006

define the label for pfmt() and lfmt(). —
setlabel, 2016

def_prog_mode — save/restore terminal
modes, 508

def_shell_mode — save/restore terminal
modes, 508

delays output — delay_output, 509

delay_output — delays output, 509

delch — remove a character, 510

delete — data base subroutines, 499

delete a window — delwin, 515

delete thread-specific data key —
pthread_key_delete, 1725

deleteln — remove a line, 513
 delwin — delete a window, 515
 del_curterm — free space pointed to by terminal, 511
 demangle — decode a C++ encoded symbol name, 516
 derwin — create a new window or subwindow, 518
 descriptions of XFN status codes — xfn_status_codes, 2548
 des_crypt — fast DES encryption, 520
 DES_FAILED — encryption failed, 520
 des_crypt — fast DES encryption, 520
 detach a name from a STREAMS-based file descriptor — fdetach, 721
 detach a thread — pthread_detach, 1711
 determine conversion object status — mbsinit, 1381
 determine insert/delete character/line capability
 — has_ic, 1119
 — has_il, 1119
 device id interfaces for a user environment
 — devid_compare, 522
 — devid_deviceid_to_nmlist, 522
 — devid_free, 522
 — devid_free_nmlist, 522
 — devid_get, 522
 — devid_get_minor_name, 522
 — devid_sizeof, 522
 device number
 manage — makedev, major, minor, 1359
 devid_compare — device id interfaces for a user environment, 522
 devid_deviceid_to_nmlist — device id interfaces for a user environment, 522
 devid_free — device id interfaces for a user environment, 522
 devid_free_nmlist — device id interfaces for a user environment, 522
 devid_get — device id interfaces for a user environment, 522
 devid_get_minor_name — device id interfaces for a user environment, 522
 devid_sizeof — device id interfaces for a user environment, 522
 dgettext — message handling function, 1082
 dial — establish an outgoing terminal line connection, 525
 difftime — computes the difference between two calendar times, 534
 directio — provide advice to file system, 553
 directories
 create, remove them in a path — mkdirp, rmdirp, 1435
 get current working directory pathname — getwd, 1106
 get pathname of current working directory — getcwd, 980
 directory operations
 — alphasort, 1935
 — scandir, 1935
 dirname — report parent directory name of file path name, 556
 disable use of certain terminal capabilities — filter, 735
 discard type-ahead characters — flushinp, 743
 display error message in standard format — pfmt, 1633
 display error message in standard format and pass to logging and monitoring services — vlfmt, 2447, 2465, 1306
 display string with video attributes
 — vidattr, 2446
 — vidputs, 2446
 — vid_attr, 2446
 — vid_puts, 2446
 dispose of connectionless LDAP pointer — cldap_close, 296
 div — compute quotient and remainder, 559
 division and remainder operations
 — div, 559
 — ldiv, 559
 dladdr — translate address to symbolic information, 564
 dlclose — close a shared object, 566
 dldump — create new file from dynamic object component of calling process, 567
 dlerror — get diagnostic information, 574
 dlopen — dynamic load information, 575
 dlopen — open a shared object, 577

dlsym — get address of symbol in shared object, 582
 DmiAddComponent — Management Interface database administration functions, 584
 DmiAddGroup — Management Interface database administration functions, 584
 DmiAddLanguage — Management Interface database administration functions, 584
 DmiAddRow — Management Interface operation functions, 589
 DmiDeleteComponent — Management Interface database administration functions, 584
 DmiDeleteGroup — Management Interface database administration functions, 584
 DmiDeleteLanguage — Management Interface database administration functions, 584
 DmiDeleteRow — Management Interface operation functions, 589
 DmiGetAttribute — Management Interface operation functions, 589
 DmiGetConfig — Management Interface initialization functions, 595
 DmiGetMultiple — Management Interface operation functions, 589
 DmiGetVersion — Management Interface initialization functions, 595
 DmiListAttributes — Management Interface listing functions, 599
 DmiListClassNames — Management Interface listing functions, 599
 DmiListComponents — Management Interface listing functions, 599
 DmiListComponentsByClass — Management Interface listing functions, 599
 DmiListGroup — Management Interface listing functions, 599
 DmiListLanguages — Management Interface listing functions, 599
 DmiOriginateEvent — Service Provider functions for components, 606
 DmiRegister — Management Interface initialization functions, 595
 DmiRegisterCi — Service Provider functions for components, 606
 DmiSetAttribute — Management Interface operation functions, 589
 DmiSetConfig — Management Interface initialization functions, 595
 DmiSetMultiple — Management Interface operation functions, 589
 DmiUnregister — Management Interface initialization functions, 595
 DmiUnRegisterCi — Service Provider functions for components, 606
 dn_comp — resolver routines, 1831
 dn_expand — resolver routines, 1831
 doconfig — execute a configuration script, 609
 door_bind — bind or unbind the current thread with the door server pool, 612
 door_call — invoke the function associated with a door descriptor, 615
 door_create — create a door descriptor, 618
 door_cred — return credential information associated with the client, 620
 door_info — return information associated with a door descriptor, 621
 door_return — return from a door invocation, 623
 door_revoke — revoke access to a door descriptor, 624
 door_server_create — specify an alternative door server thread creation function, 625
 door_unbind — bind or unbind the current thread with the door server pool, 612
 double_to_decimal — decimal record from double-precision floating, 736
 douppdate — refresh windows and lines, 628
 dup2 — duplicate an open file descriptor, 632
 duplicate a window — dupwin, 633
 duplicate an open file descriptor — dup2, 632
 dupwin — duplicate a window, 633
 dynamic linking
 close a shared object — dlclose, 566
 create new file from dynamic object component — dldump, 567

get address of symbol in shared object —
dlsym, 582
get diagnostic information — dlerror, 574
open a shared object — dlopen, 577
dynamic load information — dlinfo, 575

E

des_crypt — fast DES encryption, 520
echo — enable/disable terminal echo, 634
echochar — add a single-byte character and
refresh window, 635
echowchar — add a wchar_t character (with
attributes) to a curses window
and advance cursor, 368
echo_wchar — add a complex character and
refresh window, 636
econvert — convert number to ASCII, 637
ecvt — convert number to ASCII, 637
edata — last location in program, 694
Electronic Code Book
fast ECB encryption — ecb_crypt, 520
elf — object file access library, 650
get entries from name list — nlist, 1555
elf32_fsize — return the size of an object file
type, 641
elf32_getehdr — retrieve class-dependent
object file header, 642
elf32_getphdr — retrieve class-dependent
program header table, 644
elf32_getshdr — retrieve class-dependent
section header, 646
elf32_newehdr — retrieve class-dependent
object file header, 642
elf32_newphdr — retrieve class-dependent
program header table, 644
elf32_xlatetof — class-dependent data
translation, 648
elf32_xlatetom — class-dependent data
translation, 648
elf64_fsize — return the size of an object file
type, 641
elf64_getehdr — retrieve class-dependent
object file header, 642
elf64_getphdr — retrieve class-dependent
program header table, 644
elf64_getshdr — retrieve class-dependent
section header, 646

elf64_newehdr — retrieve class-dependent
object file header, 642
elf64_newphdr — retrieve class-dependent
program header table, 644
elf64_xlatetof — class-dependent data
translation, 648
elf64_xlatetom — class-dependent data
translation, 648
elf_begin — process ELF object files, 656
elf_cntl — control an elf file descriptor, 662
elf_end — process ELF object files, 656
elf_errmsg — error handling, 664
elf_errno — error handling, 664
elf_fill — set fill byte, 666
elf_flagdata — manipulate flags, 667
elf_flagehdr — manipulate flags, 667
elf_flagelf — manipulate flags, 667
elf_flagphdr — manipulate flags, 667
elf_flagphdr — manipulate flags, 667
elf_flagshdr — manipulate flags, 667
elf_getarhdr — retrieve archive member
header, 669
elf_getarsym — retrieve archive symbol
table, 671
elf_getbase — get the base offset for an object
file, 672
elf_getdata — get section data, 673
elf_getident — retrieve file identification
data, 678
elf_getscn — get section information, 680
elf_hash — compute hash value, 682
elf_kind — determine file type, 683
elf_memory — process ELF object files, 656
elf_ndxscn — get section information, 680
elf_newdata — get section data, 673
elf_newscn — get section information, 680
elf_next — process ELF object files, 656
elf_nextscn — get section information, 680
elf_rand — process ELF object files, 656
elf_rawdata — get section data, 673
elf_rawfile — retrieve uninterpreted file
contents, 684
elf_strptr — make a string pointer, 686
elf_update — update an ELF descriptor, 687
elf_version — coordinate ELF library and
application versions, 691
emulate the termcap database

- tgetent, 2230
- tgetflag, 2230
- tgetnum, 2230
- tgetstr, 2230
- tgoto, 2230

enable or disable cancellation —
 pthread_setcancelstate, 1760

enable/disable half-delay mode —
 halfdelay, 1118

enable/disable hardware insert-character and
delete-character features —
 idcok, 1138

enable/disable keypad handling —
 keypad, 1195

enable/disable meta keys — meta, 1434

enable/disable newline control
 — nl, 1553
 — nonl, 1553

enable/disable terminal echo
 — echo, 634
 — noecho, 634

enabling or disabling cancellation —
 pthread_setcancelstate, 1760

encryption
 determine whether a buffer of characters is
 encrypted — isencrypt, 1174

encryption functions
 password and file — crypt, 347

encryption, fast
 Cipher Block Chaining — cbc_crypt, 520
 Data Encryption Standard —
 des_crypt, 520
 Electronic Code Book — ecb_crypt, 520

end — last location in program, 694

endac — get audit control file information, 960

endauclass — close audit_class database
 file, 962

endauevent — close audit_event database
 file, 967

endauuser — get audit_user database
 entry, 970

endgrent — get group entry from
 database, 994

endpwent — get password entry from user
 database, 1047

endservent — get service entry, 1060

endspent — get shadow password database
 entry, 1075

endusershell() — function, 1092

endutent — access utmp file entry, 1093

endutxent — access utmpx file entry, 1096

endwin — restore initial terminal
 environment, 704

environment name
 return value — getenv, 990

environment variables
 change or add value — putenv, 1772

erase — clear a window, 301, 1638

erasechar — return current ERASE or KILL
 characters, 705

erasewchar — return current ERASE or KILL
 characters, 705

erf — error and complementary error
 functions, 706

erfc — error and complementary error
 functions, 706

error and complementary error functions
 — erf, 706
 — erfc, 706

error messages
 get string — strerror, 2096

error messages, system
 print — perror, 1632

etext — last location in program, 694

Ethernet address mapping operations
 — ethers, 707

ethers — Ethernet address mapping
 operations, 707

EUC character bytes
 — euclen, 709

EUC characters
 convert a string of EUC characters from
 the stream to Process Code —
 getwfs, 1109
 convert a string of Process Code characters
 to EUC characters and put it
 on a stream — putwfs, 1778

EUC codeset, get information
 — getwidth, 1107

EUC codesets, get information
 — cset, 349
 — csetcol, 349
 — csetlen, 349
 — csetno, 349
 — wcsetno, 349

EUC display width
 — euccol, 709
 — eucscol, 709
 euccol — get EUC character display width, 709
 euclen — get EUC byte length, 709
 Euclidean distance function — hypot, 1129
 eucscol — get EUC string display width, 709
 Executable and Linking Format, *see* elf,
 exit — terminate process, 710
 exit program
 add routine — atexit, 224
 exp — exponential function, 711
 expm1 — computes exponential functions, 712
 exponential function — exp, 711
 Extended Unix Code, *see* EUC,
 extended_to_decimal — decimal record from
 extended-precision
 floating, 736
 external data representation
 See XDR, 2524
 extract mantissa and exponent from double
 precision number —
 frexp, 925

F

fabs — absolute value function, 713
 fast DES encryption — des_crypt, 520
 cbc_crypt, 520
 DES_FAILED, 520
 des_failed, 520
 des_setparity, 520
 ecb_crypt, 520
 fattach — attach a STREAMS-based file
 descriptor to an object in the
 file system name space, 714
 __fbufsize — interfaces to stdio FILE
 structure, 716
 fclose — close a stream, 718
 fconvert — convert number to ASCII, 637
 fcvt — convert number to ASCII, 637
 fdatsync — synchronize a file's data, 720
 fdetach — detach a name from a
 STREAMS-based file
 descriptor, 721
 fdopen — associate a stream with a file
 descriptor, 723

FD_CLR — synchronous I/O
 multiplexing, 1966
 FD_ISSET — synchronous I/O
 multiplexing, 1966
 FD_SET — synchronous I/O
 multiplexing, 1966
 FD_ZERO — synchronous I/O
 multiplexing, 1966
 fetch — data base subroutines, 499
 fflush — flush a stream, 726
 ffs — find first set bit, 728
 fgetc — get a byte from a stream, 729
 fgetgrent — get group entry from file, 994
 fgetgrent_r — get group entry from file, 994
 fgetpos — get current file position
 information, 732
 fgetpwent — get password entry from a
 file, 1047
 fgetpwent_r — get password entry from a
 file, 1047
 fgetspent — get shadow password database
 entry, 1075
 fgetspent_r — get shadow password database
 entry(reentrant), 1075
 fgetwc — get a wide-character code from a
 stream, 733
 fgetwts —em convert a string of EUC
 characters from the stream to
 Process Code, 1109
 FIFO
 create a new one — mkfifo, 1437
 file descriptor
 duplicate an open one — dup2, 632
 STREAMS-based, attach to an object in file
 system name space —
 fattach, 714
 test for a STREAMS file — isastream, 1172
 file descriptors
 apply or remove advisory lock on open
 file — flock, 738
 file encryption functions
 — crypt, 347
 file name
 make a unique one — mktemp, 1439
 make a unique file name — mkstemp, 1438
 file pointer in a stream
 reposition — fsetpos, fgetpos, 929

file tree
 recursively descend — `ftw`, 936

files
 allows sections of file to be locked —
 `lockf`, 1341
 optimizing usage of files — `directio`, 553
 — `remove`, 1829
 report parent directory of file path name
 — `dirname`, 556
 search for named file in named directories
 — `pathfind`, 1629
 set a file to a specified length —
 `truncate`, 2395
 synchronize a file's in-memory state with
 that on the physical medium
 — `fsync`, 930

`file_to_decimal` — decimal record from
 character stream, 2112

`filter` — disable use of certain terminal
 capabilities, 735

filter expression for attribute search
 — `fn_search_filter_arguments`, 845
 — `fn_search_filter_assign`, 845
 — `fn_search_filter_copy`, 845
 — `fn_search_filter_create`, 845
 — `fn_search_filter_destroy`, 845
 — `fn_search_filter_expression`, 845
 — `FN_search_filter_t`, 845

find a wide-character in memory —
 `wmemchr`, 2511

find a wide-character substring — `wcsstr`, 2489

find pathname of a terminal
 — `ttyname`, 2431
 — `ttyname_r`, 2431

`firstkey` — data base subroutines, 499

`flash` — activate audio-visual alarm, 240

`__flbf` — interfaces to stdio FILE structure, 716

floating-point number
 convert to string — `ecvt`, 639

floating-point number, determine type
 — `finite`, 1177
 — `fpclass`, 1177
 — `isnan`, 1177
 — `isnand`, 1177
 — `isnanf`, 1177
 — `unordered`, 1177

floating-point remainder value function —
 `fmod`, 744

`flock` — apply or remove an advisory lock on
 an open file, 738

`flockfile` — acquire and release stream
 lock, 740

`floor` — floor function, 742

floor function — `floor`, 742

flush a stream — `fflush`, 726

flush non-transmitted output data, non-read
 input data or both —
 `tcflush`, 2167

flush output in tty on interrupt —
 `inrflush`, 1166

`flushinp` — discard type-ahead characters, 743

`_flushlbf` — interfaces to stdio FILE
 structure, 716

`fmod` — floating-point remainder value
 function, 744

`fmsg` — display a message on stderr or
 system console, 745

`fnmatch` — match filename or path name, 833

FNS, *see* `FN_composite_name_t`
 component names spanning multiple
 naming systems,
 `fn_attr_bind` — bind a reference to a name
 and associate attributes with
 named object, 752
 `fn_attr_create_subcontext` — create
 subcontext and associate
 attributes, 754
 `fn_attr_ext_search` — search for names
 whose attributes satisfy
 filter, 755
 `fn_attr_search` — search for atomic name
 with specified attributes in
 single context, 784
 `fn_ctx_equivalent_name` — construct
 equivalent name in same
 context, 809
 `fn_ext_searchlist_destroy` — search for
 names whose attributes
 satisfy filter, 755
 `fn_ext_searchlist_next` — search for names
 whose attributes satisfy
 filter, 755
 `FN_ext_searchlist_t` — search for names
 whose attributes satisfy
 filter, 755

fn_searchlist_destroy — terminate search for atomic name with specified attributes in single context, 784
 fn_searchlist_next — search for next atomic name with specified attributes in single context, 784
 FN_searchlist_t — search for atomic name with specified attributes in single context, 784
 FN_search_control_t — options for attribute search, 842

FNS
 FN_search_filter_t — filter expression for attribute search, 845
 fn_attribute_add — an XFN attribute, 769
 fn_attribute_assign — an XFN attribute, 769
 fn_attribute_copy — an XFN attribute, 769
 fn_attribute_create — an XFN attribute, 769
 fn_attribute_destroy — an XFN attribute, 769
 fn_attribute_first — an XFN attribute, 769
 fn_attribute_identifier — an XFN attribute, 769
 fn_attribute_next — an XFN attribute, 769
 fn_attribute_remove — an XFN attribute, 769
 fn_attribute_syntax — an XFN attribute, 769
 FN_attribute_t — an XFN attribute, 769
 fn_attribute_valuecount — an XFN attribute, 769
 fn_attrmodlist_add — a list of attribute modifications, 775
 fn_attrmodlist_assign — a list of attribute modifications, 775
 fn_attrmodlist_copy — a list of attribute modifications, 775
 fn_attrmodlist_count — a list of attribute modifications, 775
 fn_attrmodlist_create — a list of attribute modifications, 775
 fn_attrmodlist_destroy — a list of attribute modifications, 775
 fn_attrmodlist_first — a list of attribute modifications, 775
 fn_attrmodlist_next — a list of attribute modifications, 775
 FN_attrmodlist_t — a list of attribute modifications, 775
 fn_attrset_add — a set of XFN attributes, 789
 fn_attrset_assign — a set of XFN attributes, 789
 fn_attrset_copy — a set of XFN attributes, 789
 fn_attrset_count — a set of XFN attributes, 789
 fn_attrset_create — a set of XFN attributes, 789
 fn_attrset_destroy — a set of XFN attributes, 789
 fn_attrset_first — a set of XFN attributes, 789
 fn_attrset_get — a set of XFN attributes, 789
 fn_attrset_next — a set of XFN attributes, 789
 fn_attrset_remove — a set of XFN attributes, 789
 FN_attrset_t — a set of XFN attributes, 789
 fn_attr_bind — bind a reference to a name and associate attributes with named object, 752
 fn_attr_create_subcontext — create subcontext and associate attributes, 754
 fn_attr_ext_search — search for names whose attributes satisfy filter, 755
 fn_attr_get — return specified attribute associated with name, 762
 fn_attr_get_ids — get list of attribute identifiers, 764
 fn_attr_get_values — return values of an attribute, 766
 fn_attr_modify — modify specified attribute associated with name, 772
 fn_attr_multi_get — return multiple attributes associated with named object, 778
 fn_attr_multi_modify — modify multiple attributes associated with named object, 782
 fn_attr_search — search for atomic name with specified attributes in single context, 784
 fn_bindinglist_destroy — list the atomic names and references bound in a context, 819
 fn_bindinglist_next — list the atomic names and references bound in a context, 819
 FN_bindinglist_t — list the atomic names and references bound in a context, 819

`fn_composite_name_append_comp` — component names spanning multiple naming systems, 794
`fn_composite_name_append_name` — component names spanning multiple naming systems, 794
`fn_composite_name_assign` — component names spanning multiple naming systems, 794
`fn_composite_name_copy` — component names spanning multiple naming systems, 794
`fn_composite_name_count` — component names spanning multiple naming systems, 794
`fn_composite_name_create` — component names spanning multiple naming systems, 794
`fn_composite_name_delete_comp` — component names spanning multiple naming systems, 794
`fn_composite_name_destroy` — component names spanning multiple naming systems, 794
`fn_composite_name_first` — component names spanning multiple naming systems, 794
`fn_composite_name_from_str` — component names spanning multiple naming systems, 794
`fn_composite_name_from_string` — component names spanning multiple naming systems, 794
`fn_composite_name_insert_comp` — component names spanning multiple naming systems, 794
`fn_composite_name_insert_name` — component names spanning multiple naming systems, 794
`fn_composite_name_is_empty` — component names spanning multiple naming systems, 794
`fn_composite_name_is_equal` — component names spanning multiple naming systems, 794
`fn_composite_name_is_prefix` — component names spanning multiple naming systems, 794
`fn_composite_name_is_suffix` — component names spanning multiple naming systems, 794
`fn_composite_name_last` — component names spanning multiple naming systems, 794
`fn_composite_name_next` — component names spanning multiple naming systems, 794
`fn_composite_name_prefix` — component names spanning multiple naming systems, 794
`fn_composite_name_prepend_comp` — component names spanning multiple naming systems, 794
`fn_composite_name_prepend_name` — component names spanning multiple naming systems, 794
`fn_composite_name_prev` — component names spanning multiple naming systems, 794
`fn_composite_name_suffix` — component names spanning multiple naming systems, 794
`FN_composite_name_t` — component names spanning multiple naming systems, 794
`fn_compound_name_append_comp` — an XFN compound name, 799
`fn_compound_name_assign` — an XFN compound name, 799
`fn_compound_name_copy` — an XFN compound name, 799
`fn_compound_name_count` — an XFN compound name, 799
`fn_compound_name_delete_all` — an XFN compound name, 799
`fn_compound_name_delete_comp` — an XFN compound name, 799
`fn_compound_name_destroy` — an XFN compound name, 799
`fn_compound_name_first` — an XFN compound name, 799
`fn_compound_name_from_syntax_attrs` — an XFN compound name, 799
`fn_compound_name_get_syntax_attrs` — an XFN compound name, 799

fn_compound_name_insert_comp — an XFN compound name, 799
 fn_compound_name_is_empty — an XFN compound name, 799
 fn_compound_name_is_equal — an XFN compound name, 799
 fn_compound_name_is_prefix — an XFN compound name, 799
 fn_compound_name_is_suffix — an XFN compound name, 799
 fn_compound_name_last — an XFN compound name, 799
 fn_compound_name_next — an XFN compound name, 799
 fn_compound_name_prefix — an XFN compound name, 799
 fn_compound_name_prepend_comp — an XFN compound name, 799
 fn_compound_name_prev — an XFN compound name, 799
 fn_compound_name_suffix — an XFN compound name, 799
 FN_compound_name_t — an XFN compound name, 799
 fn_ctx_bind — bind a reference to a name, 804
 fn_ctx_equivalent_name — construct equivalent name in same context, 809
 fn_ctx_handle_from_initial — return a handle to the Initial Context, 815
 fn_ctx_handle_from_ref — construct a handle to a context object using the given reference, 817
 fn_ctx_list_bindings — list the atomic names and references bound in a context, 819
 fn_ctx_list_names — list the atomic names bound in a context, 821
 fn_ctx_lookup_link — look up the link reference bound to a name, 825
 fn_ctx_rename — rename the name of a binding, 826
 FN_ctx_t — an XFN context, 829
 fn_ext_searchlist_destroy — search for names whose attributes satisfy filter, 755
 fn_ext_searchlist_next — search for names whose attributes satisfy filter, 755
 FN_ext_searchlist_t — search for names whose attributes satisfy filter, 755
 FN_identifier_t — an XFN identifier, 832
 fn_multigetlist_destroy — return multiple attributes associated with named object, 778
 fn_multigetlist_next — return multiple attributes associated with named object, 778
 FN_multigetlist_t — return multiple attributes associated with named object, 778
 fn_namelist_destroy — list the atomic names bound in a context, 821
 fn_namelist_next — list the atomic names bound in a context, 821
 FN_namelist_t — list the atomic names bound in a context, 821
 fn_ref_addrcount — an XFN reference, 839
 fn_ref_addr_assign — an address in an XFN reference, 835
 fn_ref_addr_copy — an address in an XFN reference, 835
 fn_ref_addr_create — an address in an XFN reference, 835
 fn_ref_addr_data — an address in an XFN reference, 835
 fn_ref_addr_description — an address in an XFN reference, 835
 fn_ref_addr_destroy — an address in an XFN reference, 835
 fn_ref_addr_length — an address in an XFN reference, 835
 FN_ref_addr_t — an address in an XFN reference, 835
 fn_ref_addr_type — an address in an XFN reference, 835
 fn_ref_append_addr — an XFN reference, 839
 fn_ref_assign — an XFN reference, 839
 fn_ref_copy — an XFN reference, 839
 fn_ref_create — an XFN reference, 839
 fn_ref_create_link — an XFN reference, 839
 fn_ref_delete_addr — an XFN reference, 839
 fn_ref_delete_all — an XFN reference, 839

fn_ref_description — an XFN reference, 839
 fn_ref_destroy — an XFN reference, 839
 fn_ref_first — an XFN reference, 839
 fn_ref_insert_addr — an XFN reference, 839
 fn_ref_is_link — an XFN reference, 839
 fn_ref_link_name — an XFN reference, 839
 fn_ref_next — an XFN reference, 839
 fn_ref_prepend_addr — an XFN reference, 839
 FN_ref_t — an XFN reference, 839
 fn_ref_type — an XFN reference, 839
 fn_searchlist_destroy — terminate search for atomic name with specified attributes in single context, 784
 fn_searchlist_next — search for next atomic name with specified attributes in single context, 784
 FN_searchlist_t — search for atomic name with specified attributes in single context, 784
 fn_search_control_assign — options for attribute search, 842
 fn_search_control_copy — options for attribute search, 842
 fn_search_control_create — options for attribute search, 842
 fn_search_control_destroy — options for attribute search, 842
 fn_search_control_follow_links — options for attribute search, 842
 fn_search_control_max_names — options for attribute search, 842
 fn_search_control_return_attr_ids — options for attribute search, 842
 fn_search_control_return_ref — options for attribute search, 842
 fn_search_control_scope — options for attribute search, 842
 FN_search_control_t — options for attribute search, 842
 fn_search_filter_arguments — filter expression for attribute search, 845
 fn_search_filter_assign — filter expression for attribute search, 845
 fn_search_filter_copy — filter expression for attribute search, 845
 fn_search_filter_create — filter expression for attribute search, 845
 fn_search_filter_destroy — filter expression for attribute search, 845
 fn_search_filter_expression — filter expression for attribute search, 845
 FN_search_filter_t — filter expression for attribute search, 845
 BNF of Filter Expression, 846
 Extended Operations, 849
 Precedence, 847
 Relational Operators, 847
 Specification of Filter Expression, 846
 Wildcarded Strings, 848
 fn_status_advance_by_name — an XFN status object, 855
 fn_status_append_remaining_name — an XFN status object, 855
 fn_status_append_resolved_name — an XFN status object, 855
 fn_status_assign — an XFN status object, 855
 fn_status_code — an XFN status object, 855
 fn_status_copy — an XFN status object, 855
 fn_status_create — an XFN status object, 855
 fn_status_description — an XFN status object, 855
 fn_status_destroy — an XFN status object, 855
 fn_status_diagnostic_message — an XFN status object, 855
 fn_status_is_success — an XFN status object, 855
 fn_status_link_code — an XFN status object, 855
 fn_status_link_diagnostic_message — an XFN status object, 855
 fn_status_link_remaining_name — an XFN status object, 855
 fn_status_link_resolved_name — an XFN status object, 855
 fn_status_link_resolved_ref — an XFN status object, 855
 fn_status_remaining_name — an XFN status object, 855
 fn_status_resolved_name — an XFN status object, 855
 fn_status_resolved_ref — an XFN status object, 855
 fn_status_set — an XFN status object, 855

fn_status_set_code — an XFN status object, 855
 fn_status_set_diagnostic_message — an XFN status object, 855
 fn_status_set_link_code — an XFN status object, 855
 fn_status_set_link_diagnostic_message — an XFN status object, 855
 fn_status_set_link_remaining_name — an XFN status object, 855
 fn_status_set_link_resolved_name — an XFN status object, 855
 fn_status_set_link_resolved_ref — an XFN status object, 855
 fn_status_set_remaining_name — an XFN status object, 855
 fn_status_set_resolved_name — an XFN status object, 855
 fn_status_set_resolved_ref — an XFN status object, 855
 fn_status_set_success — an XFN status object, 855
 FN_status_t — an XFN status object, 855
 fn_string_assign — a character string, 859
 fn_string_bytecount — a character string, 859
 fn_string_charcount — a character string, 859
 fn_string_code_set — a character string, 859
 fn_string_compare — a character string, 859
 fn_string_compare_substring — a character string, 859
 fn_string_contents — a character string, 859
 fn_string_copy — a character string, 859
 fn_string_create — a character string, 859
 fn_string_destroy — a character string, 859
 fn_string_from_composite_name — component names spanning multiple naming systems, 794
 fn_string_from_compound_name — an XFN compound name, 799
 fn_string_from_contents — a character string, 859
 fn_string_from_str — a character string, 859
 fn_string_from_strings — a character string, 859
 fn_string_from_str_n — a character string, 859
 fn_string_from_substring — a character string, 859
 fn_string_is_empty — a character string, 859
 fn_string_next_substring — a character string, 859
 fn_string_prev_substring — a character string, 859
 fn_string_str — a character string, 859
 FN_string_t — a character string, 859
 fn_valuelist_destroy — return values of an attribute, 766
 fn_valuelist_next — return values of an attribute, 766
 FN_valuelist_t — return values of an attribute, 766
 fopen — open stream, 862, 864
 form library
 See also curses library,
 formatted input conversion — wsscanf, 2521
 formatted output conversion
 — fprintf, 1647
 — printf, 1647
 — sprintf, 1647
 — vfprintf, 1647
 — vprintf, 1647
 — vsprintf, 1647
 forms — character based forms package, 902
 forms field attributes, set and get
 — field_buffer, 877
 — field_status, 877
 — form_field_buffer, 877
 — set_field_buffer, 877
 — set_field_status, 877
 — set_max_field, 877
 forms field characteristics
 — dynamic_field_info, 879
 — field_info, 879
 — form_field_info, 879
 forms field data type validation
 — field_arg, 890
 — field_type, 890
 — form_field_validation, 890
 — set_field_type, 890
 forms field option routines
 — field_opts, 885
 — field_opts_off, 885
 — field_opts_on, 885
 — form_field_opts, 885
 — set_field_opts, 885
 forms field, off-screen data ahead or behind

- data_ahead, 868
- data_behind, 868
- form_data, 868
- forms fields, create and destroy
 - dup_field, 883
 - form_field_new, 883
 - free_field, 883
 - link_field, 883
 - new_field, 883
- forms fieldtype routines
 - form_fieldtype, 887
 - free_fieldtype, 887
 - link_fieldtype, 887
 - new_fieldtype, 887
 - set_fieldtype_arg, 887
 - set_fieldtype_choice, 887
- forms option routines
 - form_opts, 896
 - form_opts_off, 896
 - form_opts_on, 896
 - set_form_opts, 896
- forms pagination
 - form_new_page, 895
 - new_page, 895
 - set_new_page, 895
- forms window and subwindow association
 - routines
 - form_sub, 908
 - form_win, 908
 - scale_form, 908
 - set_form_sub, 908
 - set_form_win, 908
- forms window cursor, position
 - form_cursor, 867
 - pos_form_cursor, 867
- forms, application-specific routines
 - field_init, 892
 - field_term, 892
 - form_hook, 892
 - form_init, 892
 - form_term, 892
 - set_field_init, 892
 - set_field_term, 892
 - set_form_init, 892
 - set_form_term, 892
- forms, associate application data
 - field_userptr, 889
 - form_field_userptr, 889
 - form_userptr, 907
 - set_field_userptr, 889
 - set_form_userptr, 907
- forms, command processor
 - form_driver, 869
- forms, connect fields
 - field_count, 873
 - form_field, 873
 - form_fields, 873
 - move_field, 873
 - set_form_fields, 873
- forms, create and destroy
 - form_new, 894
 - free_form, 894
 - new_form, 894
- forms, format general appearance
 - field_just, 881
 - form_field_just, 881
 - set_field_just, 881
- forms, format general display attributes
 - field_back, 875
 - field_fore, 875
 - field_pad, 875
 - form_field_attributes, 875
 - set_field_back, 875
 - set_field_fore, 875
 - set_field_pad, 875
- forms, set current page and field
 - current_field, 898
 - field_index, 898
 - form_page, 898
 - set_current_field, 898
 - set_form_page, 898
- forms, write/erase from associated
 - subwindows
 - form_post, 900
 - post_form, 900
 - unpost_form, 900
- __fpending — interfaces to stdio FILE
 - structure, 716
- fpgetmask — IEEE floating-point environment
 - control, 910
- fpgetround — IEEE floating-point
 - environment control, 910
- fpgetsticky — IEEE floating-point environment
 - control, 910

fprintf — formatted output conversion, 1647, 1652
 fpsetmask — IEEE floating-point environment control, 910
 fpsetround — IEEE floating-point environment control, 910
 fpsetsticky — IEEE floating-point environment control, 910
 __fpurge — interfaces to stdio FILE structure, 716
 fputc — put a byte on a stream, 912
 fputws — put wide character string on a stream, 919
 __freachable — interfaces to stdio FILE structure, 716
 __freading — interfaces to stdio FILE structure, 716
 free — memory allocator, 267
 free space pointed to by terminal
 — del_curterm, 511
 — restartterm, 511
 — setterm, 511
 — setupterm, 511
 — set_curterm, 511
 freopen — open stream, 862, 922
 frexp — extract mantissa and exponent from double precision number, 925
 fscanf — convert formatted input, 1937
 fseek — reposition a file-position indicator in a stream, 926
 fseeko — reposition a file-position indicator in a stream, 926
 fsetpos — reposition a file pointer in a stream, 929
 fsync — synchronize a file's in-memory state with that on the physical medium, 930
 ftell — return a file offset in a stream, 932
 ftello — return a file offset in a stream, 932
 ftime — get date and time, 933
 ftruncate — set a file to a specified length, 2395
 ftw — walk a file tree, 936
 functions to manage lockfile(s) for user's mailbox
 — maillock, 1355
 — mailunlock, 1355
 — touchlock, 1355

func_to_decimal — decimal record from character function, 2112
 funlockfile — acquire and release stream lock, 740
 fwide — set stream orientation, 939
 fwprintf — print formatted wide-character output, 940
 __fwritable — interfaces to stdio FILE structure, 716
 __fwriting — interfaces to stdio FILE structure, 716
 fwscanf — convert formatted wide-character input, 947

G

gamma — log gamma function, 1311
 gamma_r — log gamma function, 1311
 gconvert — convert number to ASCII, 637
 gcvt — convert number to ASCII, 637
 general terminal interface
 — termios, 2222
 generate path name for controlling terminal
 — ctermid, 351
 — ctermid_r, 351
 generate path names matching a pattern
 — glob, 1110
 — globfree, 1110
 generic transport name-to-address translation
 — netdir, 1498
 — netdir_free, 1498
 — netdir_getbyaddr, 1498
 — netdir_getbyname, 1498
 — netdir_mergeaddr, 1498
 — netdir_options, 1498
 — netdir_perror, 1498
 — netdir_spperror, 1498
 — taddr2uaddr, 1498
 — uaddr2taddr, 1498
 get wchar_t character strings from curses terminal keyboard —
 curs_getwstr, 431, 453, 455, 457, 1060
 endservent, 1060
 getnwstr, 431
 getservbyname_r, 1060
 getservbyport, 1060

- getservbyport_r, 1060
- getservent, 1060
- getservent_r, 1060
- getwstr, 431
- innwstr, 457
- inwch, 453
- inwchnstr, 455
- inwchstr, 455
- inwstr, 457
- mvgetnwstr, 431
- mvgetwstr, 431
- mvinnwstr, 457
- mvinwch, 453
- mvinwchnstr, 455
- mvinwchstr, 455
- mvinwstr, 457
- mvwgetnwstr, 431
- mvwgetwstr, 431
- mvwinnwstr, 457
- mvwinwch, 453
- mvwinwchnstr, 455
- mvwinwchstr, 455
- mvwinwstr, 457
- setservent, 1060
- wgetnwstr, 431
- wgetwstr, 431
- winnwstr, 457
- winwch, 453
- winwchnstr, 455
- winwchstr, 455
- winwstr, 457
- get (or push back) wchar_t characters from
 - curses terminal keyboard
 - curs_getwch, 426
 - getwch, 426
 - mvgetwch, 426
 - mvwgetwch, 426
 - ungetwch, 426
 - wgetwch, 426
- get a byte from a stream
 - fgetc, 729
 - getc, 729
 - getchar, 729
 - getchar_unlocked, 729
 - getc_unlocked, 729
 - getw, 729
- get a multibyte character string from terminal
 - getnstr, 1026
 - getstr, 1026
 - mvgetnstr, 1026
 - mvgetstr, 1026
 - mvwgetnstr, 1026
 - mvwgetstr, 1026
 - wgetnstr, 1026
 - wgetstr, 1026
- get a single-byte character from terminal
 - getch, 975
 - mvgetch, 975
 - mvwgetch, 975
 - wgetch, 975
- get a synchronization object handle from a
 - synchronization object's
 - address —
 - td_ta_map_addr2sync, 2195
- get a thread's thread-specific data for
 - libthread_db library of
 - interfaces — td_thr_tsd, 2217
- get a wide character from terminal
 - get_wch, 1103
 - mvget_wch, 1103
 - mvwget_wch, 1103
 - wget_wch, 1103
- get a wide character string (with rendition)
 - from a cchar_t —
 - getcchar, 974
- get a wide character string from terminal
 - getn_wstr, 1028
 - get_wstr, 1028
 - mvgetn_wstr, 1028
 - mvget_wstr, 1028
 - mvwgetn_wstr, 1028
 - mvwget_wstr, 1028
 - wgetn_wstr, 1028
 - wget_wstr, 1028
- get a wide-character code from a stream —
 - fgetwc, 733
- get address of symbol in shared object —
 - dlsym, 582
- get and set media attributes
 - media_getattr, 1391
 - media_setattr, 1391
- get and set prioceiling attribute of mutex
 - attribute object
 - pthread_mutexattr_getprioceiling, 1727
 - pthread_mutexattr_setprioceiling, 1727

- get and set process-shared attribute
 - pthread_mutexattr_getpshared, 1732
 - pthread_mutexattr_setpshared, 1732
- get and set process-shared attribute of read-write lock attributes object
 - pthread_rwlockattr_getpshared, 1747
 - pthread_rwlockattr_setpshared, 1747
- get and set protocol attribute of mutex attribute object
 - pthread_mutexattr_getprotocol, 1729
 - pthread_mutexattr_setprotocol, 1729
- get calling thread's ID — pthread_self, 1759, 2271
- get configurable variables — confstr, 326
- get credentials of client
 - rpc_getcred, 1870
- get current file position information — fgetpos, 732
- get cursor or window coordinates
 - getbegyx, 973
 - getmaxyx, 973
 - getparyx, 973
 - getyx, 973
- get diagnostic information — dlerror, 574
- get error codes on failure
 - rpc_gss_get_error, 1872
- get execution time limits — sched_rr_get_interval, 1951
- get foreground process group ID — tcgetpgrp, 2169
- get input baud rate
 - cfgetispeed, 292
- get list of attribute identifiers — fn_attr_get_ids, 764
- get maximum data length for transmission
 - rpc_gss_max_data_length, 1878
 - rpc_gss_svc_max_data_length, 1878
- get message queue attributes — mq_getattr, 1453
- get name of signal — strsignal, 2119
- get number of bytes in a character — mblen, 1376
- get number of bytes in a character (restartable) — mbrlen, 1377
- get or set a mutex type
 - pthread_mutexattr_gettype, 1734
 - pthread_mutexattr_settype, 1734
- get or set contentionscope attribute
 - pthread_attr_getscope, 1686
 - pthread_attr_setscope, 1686
- get or set detachstate attribute
 - pthread_attr_getdetachstate, 1676
 - pthread_attr_setdetachstate, 1676
- get or set inheritsched attribute
 - pthread_attr_getinheritsched, 1680
 - pthread_attr_setinheritsched, 1680
- get or set level of concurrency
 - pthread_getconcurrency, 1715
 - pthread_setconcurrency, 1715
- get or set process scheduling priority
 - getpriority, 1039
 - setpriority, 1039
- get or set schedparam attribute
 - pthread_attr_getschedparam, 1682
 - pthread_attr_setschedparam, 1682
- get or set schedpolicy attribute
 - pthread_attr_getschedpolicy, 1684
 - pthread_attr_setschedpolicy, 1684
- get or set stackaddr attribute
 - pthread_attr_getstackaddr, 1688
 - pthread_attr_setstackaddr, 1688
- get or set stacksize attribute
 - pthread_attr_getstacksize, 1689
 - pthread_attr_setstacksize, 1689
- get or set the process-shared condition variable attributes
 - pthread_condattr_getpshared, 1696
 - pthread_condattr_setpshared, 1696
- get or set the thread guardsize attribute
 - pthread_attr_getguardsize, 1678
 - pthread_attr_setguardsize, 1678
- get output baud rate
 - cfgetospeed, 292
- get principal names at server
 - rpc_get_principal_name, 1876
- get process group ID for session leader for controlling terminal — tcgetsid, 2170
- get scheduling parameter limits
 - sched_get_priority_max, 1947
 - sched_get_priority_min, 1947
- get scheduling parameters — sched_getparam, 1946

get scheduling policy —
 sched_getscheduler, 1949
 get system load averages — getloadavg, 1009
 get the parameters associated with the
 terminal — tcgetattr, 2168
 get the trace attributes from a tnftcl handle —
 tnftcl_trace_attrs_get, 2327
 get thread information in libthread_db library
 of interfaces —
 td_thr_get_info, 2208
 get wide character from a stream —
 getwc, 1102
 get wide character from stdin stream —
 getwchar, 1105
 getacdir — get audit control file
 information, 960
 getacflg — get audit control file
 information, 960
 getacinfo — get audit control file
 information, 960
 getacmin — get audit control file
 information, 960
 getacna — get audit control file
 information, 960
 getaclassent — get audit_class database
 entry, 962
 getaclassent_r — get audit_class database
 entry, 962
 getaclassnam — get audit_class database
 entry, 962
 getaclassnam_r — get audit_class database
 entry, 962
 getauditflags() — generate process audit
 state, 992
 getauditflagsbin() — convert audit flag
 specifications, 965
 getauditflagschar() — convert audit flag
 specifications, 965
 getauevent — get audit_event database
 entry, 967
 getauevent_r — get audit_event database
 entry, 967
 getauevnam — get audit_event database
 entry, 967
 getauevnam_r — get audit_event database
 entry, 967
 getauevnonam — get audit_event database
 entry, 967
 getauevnum — get audit_event database
 entry, 967
 getauevnum_r — get audit_event database
 entry, 967
 getauserent — get audit_user database
 entry, 970
 getauserent_r — get audit_user database
 entry, 970
 getausernam — get audit_user database
 entry, 970
 getausernam_r — get audit_user database
 entry, 970
 getbegyx — get cursor or window
 coordinates, 973
 getbkgrnd — set or get the background
 character (and rendition) of
 window using a complex
 character, 259
 getc — get a byte from a stream, 729
 getcchar — get a wide character string (with
 rendition) from a cchar_t, 974
 getch — get a single-byte character from
 terminal, 975
 getchar — get a byte from a stream, 729
 getchar_unlocked — get a byte from a
 stream, 729
 getcwd — get pathname of current working
 directory, 980
 getc_unlocked — get a byte from a stream, 729
 getdate — convert user format date and
 time, 982
 General Specifications, 985
 Internal Format Conversion, 984
 Modified Conversion Specifications, 983
 getenv — return value for environment
 name, 990
 getexecname — return pathname of
 executable, 991
 getgrent — get group entry from database, 994
 getgrent_r — get group entry from
 database, 994
 getgrgid — get group entry from database, 994
 getgrgid_r — get group entry from
 database, 994
 getgrnam — get group entry from
 database, 994

getgrnam_r — get group entry from database, 994
 gethostid — get unique identifier of current host, 1004
 gethostname — get name of current host, 1005
 gethrtime — get high resolution real time, 1007
 gethrvtime — get high resolution virtual time, 1007
 getloadavg — get system load averages, 1009
 getlogin — get login name, 1010
 getlogin_r — get login name, 1010
 getmaxyx — get cursor or window coordinates, 973
 getmntany — get mnttab file information, 1012
 getmntent — get mnttab file information, 1012
 getnstr — get a multibyte character string from terminal, 1026
 getnwstr — get wchar_t character strings from curses terminal keyboard, 431
 getn_wstr — get a wide character string from terminal, 1028
 getopt — get option letter from argument vector, 1030
 getpagesize — get system page size, 1033
 getparyx — get cursor or window coordinates, 973
 getpass — read a string of characters without echo, 1034
 getpassphrase — read a string of characters without echo, 1034
 getpeername — get name of peer connected to socket, 1036
 getpriority — get or set process scheduling priority, 1039
 getpublickey — retrieve public or secret key, 1045
 getpw — get passwd entry from UID, 1046
 getpwent — get password entry from user database, 1047
 getpwent_r — get password entry from user database, 1047
 getpwnam — get password entry from user database, 1047
 getpwnam_r — get password entry from user database, 1047
 getpwuid — get password entry from user database, 1047
 getpwuid_r — get password entry from user database, 1047
 getrusage — get information about resource utilization, 1056
 gets — get the total number of threads in a process for libthread_db — td_ta_get_nthreads, 2194
 getsecretkey — retrieve public or secret key, 1045
 getservbyname — get service entry, 1060
 getservbyname_r — get service entry, 1060
 getservbyport — get service entry, 1060
 getservbyport_r — get service entry, 1060
 getservent — get service entry, 1060
 getservent_r — get service entry, 1060
 getspent — get shadow password database entry, 1075
 getspent_r — get shadow password database entry (reentrant), 1075
 getspnam — get shadow password database entry, 1075
 getspnam_r — get shadow password database entry (reentrant), 1075
 getstr — get a multibyte character string from terminal, 1026
 getsubopt — parse suboptions from a string, 1079
 gettext — message handling function, 1082
 gettimeofday — get system's notion of current Greenwich time, 1086, 1088
 gettxt — retrieve a text string, 1090
 getusershell() — get legal user shells, 1092
 getutent — access utmp file entry, 1093
 getutid — access utmp file entry, 1093
 getutline — access utmp file entry, 1093
 getutmp — access utmpx file entry, 1096
 getutmpx — access utmpx file entry, 1096
 getutxent — access utmpx file entry, 1096
 endutxent(), 1096
 getutmp(), 1096
 getutmpx(), 1096
 getutxent(), 1096
 getutxid(), 1096
 getutxline(), 1096
 pututxline(), 1096
 setutxent(), 1096
 updwtmp(), 1096

- updwtmpx(), 1096
- utmpxname(), 1096
- getutxid — access utmpx file entry, 1096
- getutxline — access utmpx file entry, 1096
- getvfsany — get vfstab file entry, 1100
- getvfsent — get vfstab file entry, 1100
- getvfsfile — get vfstab file entry, 1100
- getvfsspec — get vfstab file entry, 1100
- getw — get a byte from a stream, 729
- getwc — get wide character from a stream, 1102
- getwch — get (or push back) wchar_t characters from curses terminal keyboard, 426
- getwchar — get wide character from stdin stream, 1105
- getwd — get current working directory pathname, 1106
- getwidth — get codeset information, 1107
- getwin — read a window from, and write a window to, a file, 1108
- getws — convert a string of EUC characters from the stream to Process Code, 1109
- getwstr — get wchar_t character strings from curses terminal keyboard, 431
- getyx — get cursor or window coordinates, 973
- get_wch — get a wide character from terminal, 1103
- get_wstr — get a wide character string from terminal, 1028
- glob — generate path names matching a pattern, 1110
- global_variables — variables used for X/Open Curses, 1115
- globfree — generate path names matching a pattern, 1110
- gmatch — shell global pattern matching, 1116
- grantpt — grant access to the slave pseudo-terminal device, 1117
- graphics interface
 - arc, 1638
 - box, 1638
 - circle, 1638
 - closepl, 1638
 - closevt, 1638
 - cont, 1638

- erase, 1638
- label, 1638
- line, 1638
- linmod, 1638
- move, 1638
- openpl, 1638
- openvt, 1638
- plot, 1638
- point, 1638
- space, 1638
- group IDs
 - set terminal foreground process group id
 - tcsetpgrp, 2181
- group IDs, supplementary
 - initialize — initgroups, 1151
- groups
 - endgrent, 994
 - fgetgrent, 994
 - fgetgrent_r, 994
 - getgrent, 994
 - getgrent_r, 994
 - getgrgid, 994
 - getgrgid_r, 994
 - getgrnam, 994
 - getgrnam_r, 994
 - setgrent, 994

H

- halfdelay — enable/disable half-delay mode, 1118
- halt system processor
 - reboot, 1798
- hash-table search routine
 - hsearch, 1124
- hasmntopt — get mnttab file information, 1012
- has_colors — manipulate color information, 283
- has_ic — determine insert/delete character/line capability, 1119
- has_il — determine insert/delete character/line capability, 1119
- have Volume Management check for media — volmgt_check, 2452
- hcreate — create hash table, 1124
- hdestroy — destroy hash table, 1124

hline — use single-byte characters (and renditions) to draw lines, 1120
 hline_set — use complex characters (and renditions) to draw lines, 1122
 host ID
 get unique identifier of current host — gethostid, 1004
 host machines, remote
 return information about users — rusers, rusers, 1927
 host name
 get name of current host — gethostname, 1005
 set name of current host — sethostname, 1005
 hsearch — hash-table search routine, 1124
 hyperbolic cosine function — cosh, 345
 hyperbolic sine function — sinh, 2058
 hyperbolic tangent function — tanh, 2158
 hypot — Euclidean distance function, 1129

I

I/O multiplexing, synchronous
 — select, 1966
 I/O package
 standard buffered I/O — stdio, 2087
 I/O, asynchronous
 cancel request — aio_cancel, 198
 file synchronization — aio_sync, 202
 retrieve return status — aio_return, 210
 I/O, requests
 list — lio_listio, 1329
 iconv — code conversion function, 1130
 iconv_close — code conversion deallocation function, 1135
 iconv_open — code conversion allocation function, 1136
 idcok — enable/disable hardware insert-character and delete-character features, 1138
 idlok — set terminal output controls, 302
 IEEE arithmetic
 convert floating-point number to string — ecvt, 639
 IEEE floating-point environment control

 — fpgetmaks, 910
 — fpgetround, 910
 — fpgetsticky, 910
 — fpsetmask, 910
 — fpsetround, 910
 — fpsetsticky, 910
 ilogb — returns an unbiased exponent, 1139
 immedok — call refresh on changes to window, 1140
 inch — return a single-byte character (with rendition), 1141
 inchnstr — retrieve a single-byte character string (with rendition), 1142
 inchstr — retrieve a single-byte character string (with rendition), 1142
 index — string operations, 1144
 inet — Internet address manipulation, 1145
 inet_addr — Internet address manipulation, 1145
 inet_lnaof — Internet address manipulation, 1145
 inet_makeaddr — Internet address manipulation, 1145
 inet_netof — Internet address manipulation, 1145
 inet_network — Internet address manipulation, 1145
 inet_ntoa — Internet address manipulation, 1145
 initgroups — initialize the supplementary group access list, 1151
 initialization function for libthread_db library of interfaces — td_init, 2182
 initialize and destroy mutex attributes object
 — pthread_mutexattr_destroy, 1737
 — pthread_mutexattr_init, 1737
 initialize and destroy read-write lock attributes object
 — pthread_rwlockattr_destroy, 1749
 — pthread_rwlockattr_init, 1749
 initialize and destroy threads attribute object
 — pthread_attr_destroy, 1690
 — pthread_attr_init, 1690
 initialize dynamic package — pthread_once, 1746
 initialize kernel statistics facility
 — kstat_close, 1214

- kstat_open, 1214
- initialize or destroy a mutex
 - pthread_mutex_destroy, 1741
 - pthread_mutex_init, 1741
- initialize or destroy a read-write lock object
 - pthread_rwlock_destroy, 1751
 - pthread_rwlock_init, 1751
- initialize or destroy condition variable attributes object
 - pthread_condattr_destroy, 1698
 - pthread_condattr_init, 1698
- initialize or destroy condition variables
 - pthread_cond_destroy, 1700
 - pthread_cond_init, 1700
- initialize the LDAP library and open a connection to an LDAP server
 - ldap_init, 1288
 - ldap_open, 1288
- initscr — screen initialization functions, 1152
- initstate — pseudorandom number functions, 1784
- init_color — manipulate color information, 283
- init_pair — manipulate color information, 283
- innstr — retrieve a multibyte character string (without rendition), 1153
- innwstr — get a string of wchar_t characters from a curses window, 457, 1155
- input conversion
 - convert from wchar_t string — wscanf, 2521
- input/output package
 - standard buffered I/O — stdio, 2087
- insch — insert a character, 1157
- insdelln — insert/delete lines to/from the window, 1158
- insert a wchar_t character before the character under the cursor in a curses window — curs_inswch, 450, 451
 - insnwstr, 451
 - inswch, 450
 - inswstr, 451
 - mvinsnwstr, 451
 - mvinswch, 450
 - mvinswstr, 451
 - mvwinsnwstr, 451
 - mvwinswch, 450
- mvwinswstr, 451
- winsnwstr, 451
- winswch, 450
- winswstr, 451
- insert a character
 - insch, 1157
 - mvinsch, 1157
 - mvwinsch, 1157
 - winsch, 1157
- insert a complex character
 - ins_wch, 1165
 - mvins_wch, 1165
 - mvwins_wch, 1165
 - wins_wch, 1165
- insert a line in a window
 - insertln, 1159
 - winsertln, 1159
- insert a multibyte character string
 - insnstr, 1160
 - insstr, 1160
 - mvinsnstr, 1160
 - mvinsstr, 1160
 - mvwinsnstr, 1160
 - mvwinsstr, 1160
 - winsnstr, 1160
 - winsstr, 1160
- insert a wide character string
 - ins_nwstr, 1162
 - ins_wstr, 1162
 - mvins_nwstr, 1162
 - mvins_wstr, 1162
 - mvwins_nstr, 1162
 - mvwins_nwstr, 1162
 - wins_nwstr, 1162
 - wins_wstr, 1162
- insert/delete lines to/from the window
 - insdelln, 1158
 - winsdelln, 1158
- insertln — insert a line in a window, 1159
- insnstr — insert a multibyte character string, 1160
- insnwstr — insert wchar_t string before character under the cursor in a curses window, 451
- insque — insert element to a queue, 1164
- insstr — insert a multibyte character string, 1160

instr — retrieve a multibyte character string (without rendition), 1153
inswch — insert a `wchar_t` character before the character under the cursor in a curses window, 450
inswstr — insert `wchar_t` string before character under the cursor in a curses window, 451
ins_nwstr — insert a wide character string, 1162
ins_wch — insert a complex character, 1165
ins_wstr — insert a wide character string, 1162
interfaces for direct probe and process control for another process
 — `tnfctl_continue`, 2310
 — `tnfctl_exec_open`, 2310
 — `tnfctl_pid_open`, 2310
interfaces in `libthread_db` that target process memory access
 — `ps_pread`, 1671
 — `ps_pwrite`, 1671
 — `ps_pread`, 1671
 — `ps_ptwrite`, 1671
interfaces to query and to change the state of a probe
 — `tnfctl_probe_connect`, 2320
 — `tnfctl_probe_disable`, 2320
 — `tnfctl_probe_disconnect_all`, 2320
 — `tnfctl_probe_enable`, 2320
 — `tnfctl_probe_state_get`, 2320
 — `tnfctl_probe_trace`, 2320
 — `tnfctl_probe_untrace`, 2320
interfaces to `stdio` FILE structure
 — `__fbufsize`, 716
 — `__flbf`, 716
 — `__flushlbf`, 716
 — `__fpending`, 716
 — `__fpurge`, 716
 — `__freadable`, 716
 — `__freading`, 716
 — `__fwritable`, 716
 — `__fwriting`, 716
Internet address manipulation — `inet`, 1145
 `inet_addr`, 1145
 `inet_lnaof`, 1145
 `inet_makeaddr`, 1145
 `inet_netof`, 1145
 `inet_network`, 1145
 `inet_ntoa`, 1145
Interprocess Communication
 create a new FIFO — `mkfifo`, 1437
intrflush — flush output in tty on interrupt, 1166
introduction and overview of X/Open Curses
 — curses, 407
inverse hyperbolic functions
 — `acosh`, 180
 — `asinh`, 180
 — `atanh`, 180
invoke isa-specific executable — `isaexec`, 1170
invoke the function associated with a door descriptor — `door_call`, 615
inwch — get a `wchar_t` character and its attributes from a curses window, 453
inwchnstr — get a string of `wchar_t` characters (and attributes) from a curses window, 455
inwchstr — get a string of `wchar_t` characters (and attributes) from a curses window, 455
inwstr — get a string of `wchar_t` characters from a curses window, 457, 1155
in_wch — retrieve a complex character (with rendition), 1167
in_wchnstr — retrieve complex character string (with rendition), 1168
in_wchstr — retrieve complex character string (with rendition), 1168
isaexec — invoke isa-specific executable, 1170
isalnum — character handling, 357
isalpha — character handling, 357
isascii — character handling, 357
isatty — test for a terminal device, 1173
isdigit — character handling, 357
isencrypt — determine whether a buffer of characters is encrypted, 1174
isendwin — restore initial terminal environment, 704
isenglish — wide-character code classification functions, 1180
isgraph — character handling, 357
isideogram — wide-character code classification functions, 1180

- islower — character handling, 357
- isnan — test for NaN, 1179
- isnumber — wide-character code classification functions, 1180
- isphonogram — wide-character code classification functions, 1180
- isprint — character handling, 357
- ispunct — character handling, 357
- isspace — character handling, 357
- isspecial — wide-character code classification functions, 1180
- isupper — character handling, 357
- iswalnum — wide-character code classification functions, 1180
- iswalpha — wide-character code classification functions, 1180
- iswascii — wide-character code classification functions, 1180
- iswcntrl — wide-character code classification functions, 1180
- iswctype — test character for specified class, 1182
- iswdigit — wide-character code classification functions, 1180
- iswgraph — wide-character code classification functions, 1180
- iswlower — wide-character code classification functions, 1180
- iswprint — wide-character code classification functions, 1180
- iswpunct — wide-character code classification functions, 1180
- iswspace — wide-character code classification functions, 1180
- iswupper — wide-character code classification functions, 1180
- iswxdigit — wide-character code classification functions, 1180
- isxdigit — character handling, 357
- is_linetouched — control window refresh, 1175
- is_wintouched — control window refresh, 1175
- iterate over probes
 - tnftcl_probe_apply, 2317
 - tnftcl_probe_apply_ids, 2317
- iterate over the set of locks owned by a thread
 - td_thr_lockowner, 2212

- iterator functions on process handles from libthread_db library of interfaces
 - td_ta_sync_iter, 2201
 - td_ta_thr_iter, 2201
 - td_ta_tsd_iter, 2201

J

- j0 — Bessel functions of the first kind, 1184
- j1 — Bessel functions of the first kind, 1184
- jn — Bessel functions of the first kind, 1184

K

Kerberos authentication library

- kerberos, 1186
- krb_get_cred, 1186
- krb_kntoln, 1186
- krb_mk_err, 1186
- krb_mk_req, 1186
- krb_mk_safe, 1186
- krb_rd_err, 1186
- krb_rd_req, 1186
- krb_rd_safe, 1186
- krb_set_key, 1186

Kerberos authentication routines for RPC

- authkerb_getucred, 1191
- authkerb_seccreate, 1191
- kerberos_rpc, 1191
- svc_kerb_reg, 1191

Kerberos authentication routines via network stream sockets

- krb_net_read, 1200
- krb_net_write, 1200
- krb_recauth, 1200
- krb_sendauth, 1200

Kerberos ticket cache file name

- krb_set_tkt_string, 1204

Kerberos utility routines

- krb_get_admhst, 1197
- krb_get_krbhst, 1197
- krb_get_lrealm, 1197
- krb_get_phost, 1197
- krb_realmofhost, 1197

kernel virtual memory functions

- copy data from kernel image or running system — `kvm_read`, `kvm_kread`, `kvm_uread`, 1224
- get invocation argument for process — `kvm_getcmd`, 1216
- get entries from kernel symbol table — `kvm_nlist`, 1220
- `kstat` — kernel statistics facility, 1205
- `kstat_chain_update` — update the `kstat` header chain, 1212
- `kstat_close` — initialize kernel statistics facility, 1214
- `kstat_data_lookup` — find a `kstat` by name, 1213
- `kstat_lookup` — find a `kstat` by name, 1213
- `kstat_open` — initialize kernel statistics facility, 1214
- `kstat_read` — read or write `kstat` data, 1215
- `kstat_write` — read or write `kstat` data, 1215
- specify a kernel to examine — `kvm_open`, `kvm_close`, 1221
- `keyname` — return character string used as key name, 1194
- `keypad` — enable/disable keypad handling, 1195
- `key_name` — return character string used as key name, 1194
- `killchar` — return current ERASE or KILL characters, 705
- `killpg` — send signal to a process group, 1196
- `killwchar` — return current ERASE or KILL characters, 705
- `kstat` — kernel statistics facility, 1205
- `kstat_chain_update` — update the `kstat` header chain, 1212
- `kstat_close` — initialize kernel statistics facility, 1214
- `kstat_data_lookup` — find a `kstat` by name, 1213
- `kstat_lookup` — find a `kstat` by name, 1213
- `kstat_open` — initialize kernel statistics facility, 1214
- `kstat_read` — read or write `kstat` data, 1215
- `kstat_write` — read or write `kstat` data, 1215
- `kvm_close` — specify kernel to examine, 1221

- `kvm_getcmd` — get invocation arguments for process, 1216
- `kvm_getproc` — read system process structures, 1218
- `kvm_getu` — get u-area for process, 1216
- `kvm_kread` — copy data from a kernel image or running system, 1224
- `kvm_kwrite` — copy data to a kernel image or running system, 1224
- `kvm_nextproc` — read system process structures, 1218
- `kvm_nlist` — get entries from kernel symbol table, 1220
- `kvm_open` — specify kernel to examine, 1221
- `kvm_read` — copy data from kernel image or running system, 1224
- `kvm_setproc` — read system process structures, 1218
- `kvm_uread` — copy data from a kernel image or running system, 1224
- `kvm_uwrite` — copy data to a kernel image or running system, 1224
- `kvm_write` — copy data to kernel image or running system, 1224

L

- `label` — graphics interface, 1638
- `labs` — return absolute value of long integer, 166
- language information — `nl_langinfo`, 1556
- `ldap` — Lightweight Directory Access Protocol package, 1227
 - BER Library, 1228
 - Caching, 1228
 - Connectionless Access, 1228
 - Displaying Results, 1227
 - Index, 1228
 - Search Filters, 1227
 - User Friendly Naming, 1228
- LDAP attribute remapping functions
 - `ldap_free_friendlymap`, 1274
 - `ldap_friendly_name`, 1274
- LDAP attribute value handling functions
 - `ldap_count_values`, 1281
 - `ldap_get_values`, 1281
 - `ldap_get_values_len`, 1281

- LDAP bind functions
 - ldap_bind, 1241
 - ldap_bind_s, 1241
 - ldap_sasl_bind, 1241
 - ldap_sasl_bind_s, 1241
 - ldap_set_rebind_proc, 1241
 - ldap_simple_bind, 1241
 - ldap_simple_bind_s, 1241
 - ldap_unbind, 1241
 - ldap_unbind_s, 1241
- LDAP character set translation functions
 - ldap_8859_to_t61, 1246
 - ldap_enable_translation, 1246
 - ldap_set_string_translators, 1246
 - ldap_t61_to_8859, 1246
 - ldap_translate_from_t61, 1246
 - ldap_translate_to_t61, 1246
- LDAP client caching functions
 - ldap_cache, 1244
 - ldap_destroy_cache, 1244
 - ldap_disable_cache, 1244
 - ldap_enable_cache, 1244
 - ldap_flush_cache, 1244
 - ldap_set_cache_options, 1244
 - ldap_uncache_entry, 1244
 - ldap_uncache_request, 1244
- LDAP compare operation
 - ldap_compare, 1248
 - ldap_compare_ext, 1248
 - ldap_compare_ext_s, 1248
 - ldap_compare_s, 1248
- LDAP connectionless communication
 - preparation —
 - cldap_open, 297
- LDAP control disposal
 - ldap_controls_free, 1250
 - ldap_control_free, 1250
- LDAP delete operation
 - ldap_delete, 1251
 - ldap_delete_ext, 1251
 - ldap_delete_ext_s, 1251
 - ldap_delete_s, 1251
- LDAP display template functions
 - ldap_disptmpl, 1253
 - ldap_first_disptmpl, 1253
 - ldap_first_tmplcol, 1253
 - ldap_first_tmplrow, 1253
 - ldap_free_templates, 1253
 - ldap_init_templates, 1253
 - ldap_init_templates_buf, 1253
 - ldap_next_disptmpl, 1253
 - ldap_next_tmplcol, 1253
 - ldap_next_tmplrow, 1253
 - ldap_oc2template, 1253
 - ldap_tmplattrs, 1253
- LDAP DN handling functions
 - ldap_dn2ufn, 1276
 - ldap_dns_to_dn, 1276
 - ldap_explode_dn, 1276
 - ldap_explode_dns, 1276
 - ldap_get_dn, 1276
 - ldap_is_dns_dn, 1276
- LDAP entry display functions
 - ldap_entry2text, 1260
 - ldap_entry2text_search, 1260
 - ldap_vals2text, 1260
- LDAP entry modification functions
 - ldap_modify, 1283
 - ldap_modify_ext, 1283
 - ldap_modify_ext_s, 1283
 - ldap_modify_s, 1283
- LDAP entry parsing and counting functions
 - ldap_count_entries, 1270
 - ldap_count_references, 1270
 - ldap_first_entry, 1270
 - ldap_first_reference, 1270
 - ldap_next_entry, 1270
- LDAP entry sorting functions
 - ldap_sort, 1298
 - ldap_sort_entries, 1298
 - ldap_sort_strcasecmp, 1298
 - ldap_sort_values, 1298
- LDAP filter generating functions
 - ldap_build_filter, 1278
 - ldap_getfilter, 1278
 - ldap_getfilter_free, 1278
 - ldap_getfirstfilter, 1278
 - ldap_getnextfilter, 1278
 - ldap_init_getfilter, 1278
 - ldap_init_getfilter_buf, 1278
- LDAP message processing functions
 - ldap_count_message, 1272
 - ldap_first_message, 1272
 - ldap_msgtype, 1272
 - ldap_next_message, 1272

- LDAP message result parser
 - ldap_parse_extended_result, 1290
 - ldap_parse_result, 1290
 - ldap_parse_sasl_bind_result, 1290
- LDAP protocol error handling functions
 - ldap_err2string, 1264
 - ldap_errlist, 1264
 - ldap_error, 1264
 - ldap_perror, 1264
 - ldap_result2error, 1264
 - ld_errno, 1264
- LDAP search operations
 - ldap_search, 1293
 - ldap_search_ext, 1293
 - ldap_search_ext_s, 1293
 - ldap_search_s, 1293
 - ldap_search_st, 1293
- LDAP search preference configuration routines
 - ldap_first_searchobj, 1296
 - ldap_free_searchprefs, 1296
 - ldap_init_searchprefs, 1296
 - ldap_init_searchprefs_buf, 1296
 - ldap_next_searchobj, 1296
 - ldap_searchprefs, 1296
- LDAP Uniform Resource Locator functions
 - ldap_dns_to_url, 1302
 - ldap_dn_to_url, 1302
 - ldap_free_urldesc, 1302
 - ldap_is_ldap_url, 1302
 - ldap_url, 1302
 - ldap_url_parse, 1302
 - ldap_url_search, 1302
 - ldap_url_search_s, 1302
 - ldap_url_search_st, 1302
- LDAP user friendly search functions
 - ldap_ufn, 1300
 - ldap_ufn_search_c, 1300
 - ldap_ufn_search_ct, 1300
 - ldap_ufn_search_s, 1300
 - ldap_ufn_setfilter, 1300
 - ldap_ufn_setprefix, 1300
 - ldap_ufn_timeout, 1300
- ldap_8859_to_t61 — LDAP character set translation functions, 1246
- ldap_abandon — abandon an LDAP operation in progress, 1238
- ldap_add — perform an LDAP add operation, 1239
- ldap_add_ext — perform an LDAP add operation, 1239
- ldap_add_ext_s — perform an LDAP add operation, 1239
- ldap_add_s — perform an LDAP add operation, 1239
- ldap_bind — LDAP bind functions, 1241
 - General Authentication, 1242
 - Re-Binding While Following Referral, 1242
 - Simple Authentication, 1241
 - Unbinding, 1242
- ldap_bind_s — LDAP bind functions, 1241
- ldap_build_filter — LDAP filter generating functions, 1278
- ldap_cache — LDAP client caching functions, 1244
- ldap_compare — LDAP compare operation, 1248
- ldap_compare_ext — LDAP compare operation, 1248
- ldap_compare_ext_s — LDAP compare operation, 1248
- ldap_compare_s — LDAP compare operation, 1248
- ldap_controls_free — LDAP control disposal, 1250
- ldap_control_free — LDAP control disposal, 1250
- ldap_count_entries — LDAP entry parsing and counting functions, 1270
- ldap_count_message — LDAP message processing functions, 1272
- ldap_count_references — LDAP entry parsing and counting functions, 1270
- ldap_count_values — LDAP attribute value handling functions, 1281
- ldap_delete — LDAP delete operation, 1251
- ldap_delete_ext — LDAP delete operation, 1251
- ldap_delete_ext_s — LDAP delete operation, 1251
- ldap_delete_s — LDAP delete operation, 1251
- ldap_destroy_cache — LDAP client caching functions, 1244

ldap_disable_cache — LDAP client caching functions, 1244
 ldap_disptmpl — LDAP display template functions, 1253
 DISPTMPL Structure Elements, 1255
 Syntax IDs, 1257
 TMPLITEM Structure Elements, 1257
 ldap_dn2ufn — LDAP DN handling functions, 1276
 ldap_dns_to_dn — LDAP DN handling functions, 1276
 ldap_dns_to_url — LDAP Uniform Resource Locator functions, 1302
 ldap_dn_to_url — LDAP Uniform Resource Locator functions, 1302
 ldap_enable_cache — LDAP client caching functions, 1244
 ldap_enable_translation — LDAP character set translation functions, 1246
 ldap_entry2text — LDAP entry display functions, 1260
 ldap_entry2text_search — LDAP entry display functions, 1260
 ldap_err2string — LDAP protocol error handling functions, 1264
 ldap_errlist — LDAP protocol error handling functions, 1264
 ldap_error — LDAP protocol error handling functions, 1264
 ldap_explode_dn — LDAP DN handling functions, 1276
 ldap_explode_dns — LDAP DN handling functions, 1276
 ldap_first_attribute — step through LDAP entry attributes, 1268
 ldap_first_disptmpl — LDAP display template functions, 1253
 ldap_first_entry — LDAP entry parsing and counting functions, 1270
 ldap_first_message — LDAP message processing functions, 1272
 ldap_first_reference — LDAP entry parsing and counting functions, 1270
 ldap_first_searchobj — LDAP search preference configuration routines, 1296
 ldap_first_tmplcol — LDAP display template functions, 1253
 ldap_first_tmplrow — LDAP display template functions, 1253
 ldap_flush_cache — LDAP client caching functions, 1244
 ldap_free_friendlymap — LDAP attribute remapping functions, 1274
 ldap_free_searchprefs — LDAP search preference configuration routines, 1296
 ldap_free_templates — LDAP display template functions, 1253
 ldap_free_urldesc — LDAP Uniform Resource Locator functions, 1302
 ldap_friendly_name — LDAP attribute remapping functions, 1274
 ldap_getfilter — LDAP filter generating functions, 1278
 ldap_getfilter_free — LDAP filter generating functions, 1278
 ldap_getfirstfilter — LDAP filter generating functions, 1278
 ldap_getnextfilter — LDAP filter generating functions, 1278
 ldap_get_dn — LDAP DN handling functions, 1276
 ldap_get_values — LDAP attribute value handling functions, 1281
 ldap_get_values_len — LDAP attribute value handling functions, 1281
 ldap_init — initialize the LDAP library and open a connection to an LDAP server, 1288
 ldap_init_getfilter — LDAP filter generating functions, 1278
 ldap_init_getfilter_buf — LDAP filter generating functions, 1278
 ldap_init_searchprefs — LDAP search preference configuration routines, 1296
 ldap_init_searchprefs_buf — LDAP search preference configuration routines, 1296
 ldap_init_templates — LDAP display template functions, 1253
 ldap_init_templates_buf — LDAP display template functions, 1253

ldap_is_dns_dn — LDAP DN handling functions, 1276
 ldap_is_ldap_url — LDAP Uniform Resource Locator functions, 1302
 ldap_modify — LDAP entry modification functions, 1283
 ldap_modify_ext — LDAP entry modification functions, 1283
 ldap_modify_ext_s — LDAP entry modification functions, 1283
 ldap_modify_s — LDAP entry modification functions, 1283
 ldap_modrdn — modify LDAP entry RDN, 1286
 ldap_modrdn2 — modify LDAP entry RDN, 1286
 ldap_modrdn2_s — modify LDAP entry RDN, 1286
 ldap_modrdn_s — modify LDAP entry RDN, 1286
 ldap_msgtype — LDAP message processing functions, 1272
 ldap_next_attribute — step through LDAP entry attributes, 1268
 ldap_next_disptmpl — LDAP display template functions, 1253
 ldap_next_entry — LDAP entry parsing and counting functions, 1270
 ldap_next_message — LDAP message processing functions, 1272
 ldap_next_searchobj — LDAP search preference configuration routines, 1296
 ldap_next_tmplcol — LDAP display template functions, 1253
 ldap_next_tmplrow — LDAP display template functions, 1253
 ldap_oc2template — LDAP display template functions, 1253
 ldap_open — initialize the LDAP library and open a connection to an LDAP server, 1288
 ldap_parse_extended_result — LDAP message result parser, 1290
 ldap_parse_result — LDAP message result parser, 1290
 ldap_parse_sasl_bind_result — LDAP message result parser, 1290
 ldap_perror — LDAP protocol error handling functions, 1264
 ldap_rename — modify LDAP entry RDN, 1286
 ldap_rename_s — modify LDAP entry RDN, 1286
 ldap_result — wait for and return LDAP operation result, 1291
 ldap_result2error — LDAP protocol error handling functions, 1264
 ldap_sasl_bind — LDAP bind functions, 1241
 ldap_sasl_bind_s — LDAP bind functions, 1241
 ldap_search — LDAP search operations, 1293
 ldap_searchprefs — LDAP search preference configuration routines, 1296
 ldap_search_ext — LDAP search operations, 1293
 ldap_search_ext_s — LDAP search operations, 1293
 ldap_search_s — LDAP search operations, 1293
 ldap_search_st — LDAP search operations, 1293
 ldap_set_cache_options — LDAP client caching functions, 1244
 ldap_set_rebind_proc — LDAP bind functions, 1241
 ldap_set_string_translators — LDAP character set translation functions, 1246
 ldap_simple_bind — LDAP bind functions, 1241
 ldap_simple_bind_s — LDAP bind functions, 1241
 ldap_sort — LDAP entry sorting functions, 1298
 ldap_sort_entries — LDAP entry sorting functions, 1298
 ldap_sort_strcasecmp — LDAP entry sorting functions, 1298
 ldap_sort_values — LDAP entry sorting functions, 1298
 ldap_t61_to_8859 — LDAP character set translation functions, 1246
 ldap_tmplattrs — LDAP display template functions, 1253

ldap_translate_from_t61 — LDAP character set translation functions, 1246
 ldap_translate_to_t61 — LDAP character set translation functions, 1246
 ldap_ufn — LDAP user friendly search functions, 1300
 ldap_ufn_search_c — LDAP user friendly search functions, 1300
 ldap_ufn_search_ct — LDAP user friendly search functions, 1300
 ldap_ufn_search_s — LDAP user friendly search functions, 1300
 ldap_ufn_setfilter — LDAP user friendly search functions, 1300
 ldap_ufn_setprefix — LDAP user friendly search functions, 1300
 ldap_ufn_timeout — LDAP user friendly search functions, 1300
 ldap_unbind — LDAP bind functions, 1241
 ldap_unbind_s — LDAP bind functions, 1241
 ldap_uncache_entry — LDAP client caching functions, 1244
 ldap_uncache_request — LDAP client caching functions, 1244
 ldap_url — LDAP Uniform Resource Locator functions, 1302
 ldap_url_parse — LDAP Uniform Resource Locator functions, 1302
 ldap_url_search — LDAP Uniform Resource Locator functions, 1302
 ldap_url_search_s — LDAP Uniform Resource Locator functions, 1302
 ldap_url_search_st — LDAP Uniform Resource Locator functions, 1302
 ldap_vals2text — LDAP entry display functions, 1260
 ldexp — load exponent of a floating point number, 1305
 ldiv — compute quotient and remainder, 559
 ld_errno — LDAP protocol error handling functions, 1264
 leaveok — set terminal output controls, 302
 lfmt — display error message in standard format and pass to logging and monitoring services, 1306
 lgamma — log gamma function, 1311
 lgamma_r — log gamma function, 1311

libdevinfo — library of device information functions, 1313
 library for TNF probe control in a process or the kernel — libtnfctl, 1324
 library of device information functions — libdevinfo, 1313
 library of interfaces for monitoring and manipulating threads-related aspects of multithreaded programs — libthread_db, 1318
 library routines for client side calls

- clnt_call, 1858
- clnt_freeres, 1858
- clnt_geterr, 1858
- clnt_perrno, 1858
- clnt_perror, 1858
- clnt_sperrno, 1858
- clnt_spperror, 1858
- rpc_broadcast, 1858
- rpc_broadcast_exp, 1858
- rpc_call, 1858
- rpc_clnt_calls, 1858

 library routines for dealing with creation and manipulation of CLIENT handles

- clnt_control, 1862
- clnt_create, 1862
- clnt_create_timed, 1862
- clnt_create_vers, 1862
- clnt_create_vers_timed, 1862
- clnt_destroy, 1862
- clnt_dg_create, 1862
- clnt_pcreateerror, 1862
- clnt_raw_create, 1862
- clnt_sppcreateerror, 1862
- clnt_tli_create, 1862
- clnt_tp_create, 1862
- clnt_tp_create_timed, 1862
- clnt_vc_create, 1862
- rpc_clnt_create, 1862
- rpc_createerr, 1862

 library routines for RPC servers

- rpc_svc_calls, 1910
- svc_dg_enablecache, 1910
- svc_done, 1910
- svc_exit, 1910

- svc_fdset, 1910
- svc_freeargs, 1910
- svc_getargs, 1910
- svc_getreqset, 1910
- svc_getreq_common, 1910
- svc_getreq_poll, 1910
- svc_getrpcaller, 1910
- svc_max_pollfd, 1910
- svc_pollfd, 1910
- svc_run, 1910
- svc_sendreply, 1910
- libthread_db — library of interfaces for monitoring and manipulating threads-related aspects of multithreaded programs, 1318
- libtntctl — library for TNF probe control in a process or the kernel, 1324
- Lightweight Directory Access Protocol package — ldap, 1227
- line — graphics interface, 1638
- linear search and update routine
 - lfind, 1351
 - lsearch, 1351
- linmod — graphics interface, 1638
- lio_listio — list directed I/O, 1329
- list directed I/O — lio_listio, 1329
- list the atomic names and references bound in a context
 - fn_bindinglist_destroy, 819
 - fn_bindinglist_next, 819
 - FN_bindinglist_t, 819
 - fn_ctx_list_bindings, 819
- list the atomic names bound in a context
 - fn_ctx_list_names, 821
 - fn_namelist_destroy, 821
 - fn_namelist_next, 821
 - FN_namelist_t, 821
- listen — listen for connections on a socket, 1333
- llabs — return absolute value of long long integer, 166
- lldiv — compute quotient and remainder, 559
- lltostr — string conversion routines, 2122
- load exponent of a floating point number — ldexp, 1305
- load exponent of a radix-independent floating-point number — scalb, 1933, 1934
- locale
 - modify and query a program's locale — setlocale, 2017
- localeconv — get numeric formatting information, 1336
- lock
 - apply or remove advisory lock on open file — flock, 738
- lock address space
 - mlockall, 1444
- lock memory pages
 - mlock, 1442
- lock or attempt to lock a read-write lock object for reading
 - pthread_rwlock_rdlock, 1753
 - pthread_rwlock_tryrdlock, 1753
- lock or attempt to lock a read-write lock object for writing
 - pthread_rwlock_trywrlock, 1757
 - pthread_rwlock_wrlock, 1757
- lock or unlock a mutex
 - pthread_mutex_lock, 1743
 - pthread_mutex_trylock, 1743
 - pthread_mutex_unlock, 1743
- lockf — allows sections of file to be locked, 1341
- log — natural logarithm function, 1347
 - gamma, 1311
 - gamma_r, 1311
 - lgamma, 1311
 - lgamma_r, 1311
- log10 — base 10 logarithm function, 1345
- log1p — compute natural logarithm, 1346
- logb — radix-independent exponent, 1348
- login name
 - getlogin, 1010
 - getlogin_r, 1010
- longjmp — non-local goto, 2008, 2012
- _longjmp — non-local goto, 1349, 2008
- longname — return full terminal type name, 1350
- looks up the symbol in the symbol table of the load object in the target

- process —
 - ps_pglobal_lookup, 1670
- look up the link reference bound to a name —
 - fn_ctx_lookup_link, 825

M

- madvise — provide advice to VM system, 1353
- maillock — functions to manage lockfile(s) for user's mailbox, 1355
- mailunlock — functions to manage lockfile(s) for user's mailbox, 1355
- make modified instructions executable —
 - sync_instruction_memory, 2131
- makecontext — manipulate user contexts, 1357
- malloc — memory allocator, 267
- manage thread signals for libthread_db
 - td_thr_setsigpending, 2214
 - td_thr_sigsetmask, 2214
- manage thread-specific data
 - pthread_getspecific, 1719
 - pthread_setspecific, 1719
- Management Interface database administration functions
 - DmiAddComponent, 584
 - DmiAddGroup, 584
 - DmiAddLanguage, 584
 - DmiDeleteComponent, 584
 - DmiDeleteGroup, 584
 - DmiDeleteLanguage, 584
- Management Interface initialization functions
 - DmiGetConfig, 595
 - DmiGetVersion, 595
 - DmiRegister, 595
 - DmiSetConfig, 595
 - DmiUnregister, 595
- Management Interface listing functions
 - DmiListAttributes, 599
 - DmiListClassNames, 599
 - DmiListComponents, 599
 - DmiListComponentsByClass, 599
 - DmiListGroups, 599
 - DmiListLanguages, 599
- Management Interface operation functions
 - DmiAddRow, 589
 - DmiDeleteRow, 589
 - DmiGetAttribute, 589
 - DmiGetMultiple, 589

- DmiSetAttribute, 589
- DmiSetMultiple, 589
- manipulate sets of signals — sigsetops, 2044
 - sigaddset, 2044
 - sigdelset, 2044
 - sigemptyset, 2044
 - sigfillset, 2044
 - sigismember, 2044
- manipulate color information
 - can_change_color, 283
 - color_content, 283
 - COLOR_PAIR, 283
 - has_colors, 283
 - init_color, 283
 - init_pair, 283
 - pair_content, 283
 - PAIR_NUMBER, 283
 - start_color, 283
- manipulate soft labels
 - slk_attroff, 2062
 - slk_atron, 2062
 - slk_attrset, 2062
 - slk_attr_off, 2062
 - slk_attr_on, 2062
 - slk_attr_set, 2062
 - slk_clear, 2062
 - slk_color, 2062
 - slk_init, 2062
 - slk_label, 2062
 - slk_noutrefresh, 2062
 - slk_refresh, 2062
 - slk_restore, 2062
 - slk_set, 2062
 - slk_touch, 2062
 - slk_wset, 2062
- map a tnftcl error code to a string —
 - tnftcl_strerror, 2326
- map area of parent window to subwindow —
 - mvderwin, 1489
- map ASCII mechanism to OID
 - rpc_gss_mech_to_oid, 1874, 1880
- map ASCII qop to number
 - rpc_gss_qop_to_num, 1874, 1880
- match filename or path name — fnmatch, 833
- math library exception-handling —
 - matherr, 1369
- mathematical functions

- gamma, 1311
- gamma_r, 1311
- lgamma, 1311
- lgamma_r, 1311
- matherr — math library
 - exception-handling, 1369
- mblen — get number of bytes in a character, 1376
- mbrlen — get number of bytes in a character (restartable), 1377
- mbrtowc — convert a character to a wide-character code (restartable), 1379
- mbsinit — determine conversion object status, 1381
- mbsrtowcs — convert a character string to a wide-character string (restartable), 1382
- mbstowcs — convert a character string to a wide-character string, 1384
- mbtowc — convert a character to a wide-character code, 1385
- mctl — memory management control, 1386
- media_findname — convert a supplied name into an absolute pathname that can be used to access removable media, 1389
- media_getattr — get and set media attributes, 1391
- media_setattr — get and set media attributes, 1391
- memory — memory operations, 1396
 - optimizing usage of user mapped memory — madvise, 1353
- memory allocator — bsdmalloc, 267
 - alloca, 1360
 - calloc, 1360, 1363, 1367, 1469
 - free, 267, 1360, 1363, 1367, 1469
 - mallinfo, 1363
 - malloc, 267, 1360, 1363, 1367, 1469
 - mallopt, 1363
 - memalign, 1360
 - realloc, 267, 1360, 1363, 1367, 1469
 - valloc, 1360
- memory lock or unlock
 - calling process — plock, 1637
- memory management — mctl, 1386
 - copy a file into memory — copypart, 340
 - get system page size — getpagesize, 1033
 - lock pages in memory — mlock, 1442, 1444
 - synchronize memory with physical storage — msync, 1467
 - unlock pages in memory — munlock, 1442, 1444
- memory object, shared
 - open — shm_open, 2024
 - remove — shm_unlink, 2027
- memory operations
 - memccpy, 1396
 - memchr, 1396
 - memcmp, 1396
 - memcpy, 1396
 - memmove, 1396
 - memory, 1396
 - memset, 1396
- menu library
 - See also* curses library,
- menus — character based menus package, 1427
- menus cursor
 - menu_cursor, 1400
 - pos_menu_cursor, 1400
- menus display attributes
 - menu_attributes, 1398
 - menu_back, 1398
 - menu_fore, 1398
 - menu_grey, 1398
 - menu_pad, 1398
 - set_menu_back, 1398
 - set_menu_fore, 1398
 - set_menu_grey, 1398
 - set_menu_pad, 1398
- menus from associated subwindows, write/erase
 - menu_post, 1425
 - post_menu, 1425
 - unpost_menu, 1425
- menus item name and description
 - item_description, 1409
 - item_name, 1409
 - menu_item_name, 1409
- menus item options routines
 - item_opts, 1412
 - item_opts_off, 1412

- item_opts_on, 1412
- menu_item_opts, 1412
- set_item_opts, 1412
- menus item values, set and get
 - item_value, 1417
 - menu_item_value, 1417
 - set_item_value, 1417
- menus item, visibility
 - item_visible, 1419
 - menu_item_visible, 1419
- menus items, associate application data
 - item_userptr, 1416
 - menu_item_userptr, 1416
 - set_item_userptr, 1416
- menus items, connect and disconnect
 - item_count, 1414
 - menu_items, 1414
 - set_menu_items, 1414
- menus items, create and destroy
 - free_item, 1410
 - menu_item_new, 1410
 - new_item, 1410
- menus items, get and set
 - current_item, 1407
 - item_index, 1407
 - menu_item_current, 1407
 - set_current_item, 1407
 - set_top_row, 1407
 - top_row, 1407
- menus mark string routines
 - menu_mark, 1420
 - set_menu_mark, 1420
- menus options routines
 - menu_opts, 1422
 - menu_opts_off, 1422
 - menu_opts_on, 1422
 - set_menu_opts, 1422
- menus pattern match buffer
 - menu_pattern, 1424
 - set_menu_pattern, 1424
- menus subsystem, command processor
 - menu_driver, 1401
- menus window and subwindow association
 - routines
 - menu_sub, 1432
 - menu_win, 1432
 - scale_menu, 1432
 - set_menu_sub, 1432
 - set_menu_win, 1432
- menus, application-specific routines
 - item_init, 1405
 - item_term, 1405
 - menu_hook, 1405
 - menu_init, 1405
 - menu_term, 1405
 - set_item_init, 1405
 - set_item_term, 1405
 - set_menu_init, 1405
 - set_menu_term, 1405
- menus, associate application data
 - menu_userptr, 1431
 - set_menu_userptr, 1431
- menus, create and destroy
 - free_menu, 1421
 - menu_new, 1421
 - new_menu, 1421
- menus, rows and columns
 - menu_format, 1403
 - set_menu_format, 1403
- message catalog
 - open/catalog — catopen, catclose, 286
 - read a program message — catgets, 285
- message handling functions
 - bindtextdomain, 1082
 - dcgettext, 1082
 - dgettext, 1082
 - gettext, 1082
 - textdomain, 1082
- message queue
 - close — mq_close, 1452
 - notify process (or thread) —
 - mq_notify, 1455
 - open — mq_open, 1457
 - receive a message from —
 - mq_receive, 1461
 - remove — mq_unlink, 1466
 - send message to — mq_send, 1463
 - set attributes — mq_setattr, 1465
- messages
 - display a message on stderr or system
 - console — fntmsg, 745
 - print system error messages —
 - perror, 1632
 - system signal messages — psignal, 1667
- meta — enable/disable meta keys, 1434

mkdirp — create directories in a path, 1435
 mkfifo — create a new FIFO, 1437
 mkstemp — make a unique file name, 1438
 mktemp — make a unique file name, 1439
 mktime — converts a tm structure to a
 calendar time, 1440
 mnttab file
 — getmntany, 1012
 — getmntent, 1012
 — hasmntopt, 1012
 — putmntent, 1012
 modf — decompose floating-point
 number, 1446
 modff — decompose floating-point
 number, 1446
 modify LDAP entry RDN
 — ldap_modrdn, 1286
 — ldap_modrdn2, 1286
 — ldap_modrdn2_s, 1286
 — ldap_modrdn_s, 1286
 — ldap_rename, 1286
 — ldap_rename_s, 1286
 modify multiple attributes associated with
 named object —
 fn_attr_multi_modify, 782
 modify specified attribute associated with
 name — fn_attr_modify, 772
 modify/delete user credentials for an
 authentication service —
 pam_setcred, 1594
 monitor — prepare process execution
 profile, 1447
 move — move cursor in window, 1449, 1638
 move cursor in window
 — move, 1449
 — wmove, 1449
 move the cursor — mvcur, 1488
 move window — mvwin, 1494
 movenextch — moving the cursor by
 character, 374
 moveprevch — moving the cursor by
 character, 374
 mp — multiple precision integer
 arithmetic, 1450
 mp_gcd — multiple precision integer
 arithmetic, 1450
 mp_itom — multiple precision integer
 arithmetic, 1450
 mp_madd — multiple precision integer
 arithmetic, 1450
 mp_mcmp — multiple precision integer
 arithmetic, 1450
 mp_mdiv — multiple precision integer
 arithmetic, 1450
 mp_mfree — multiple precision integer
 arithmetic, 1450
 mp_min — multiple precision integer
 arithmetic, 1450
 mp_mout — multiple precision integer
 arithmetic, 1450
 mp_msub — multiple precision integer
 arithmetic, 1450
 mp_mtox — multiple precision integer
 arithmetic, 1450
 mp_mult — multiple precision integer
 arithmetic, 1450
 mp_pow — multiple precision integer
 arithmetic, 1450
 mp_rpow — multiple precision integer
 arithmetic, 1450
 mp_xtom — multiple precision integer
 arithmetic, 1450
 mq_close — close a message queue, 1452
 mq_getattr — get message queue
 attributes, 1453
 mq_notify — notify process (or thread) that a
 message is available on a
 queue, 1455
 mq_open — open a message queue, 1457
 mq_receive — receive a message from a
 message queue, 1461
 mq_send — send a message to a message
 queue, 1463
 mq_setattr — set/get message queue
 attributes, 1465
 mq_unlink — remove a message queue, 1466
 msync — synchronize memory with physical
 storage, 1467
 multiple precision integer arithmetic
 — mp, 1450
 — mp_gcd, 1450
 — mp_itom, 1450
 — mp_madd, 1450
 — mp_mcmp, 1450
 — mp_mdiv, 1450

- mp_mfree, 1450
- mp_min, 1450
- mp_mout, 1450
- mp_msub, 1450
- mp_mtox, 1450
- mp_mult, 1450
- mp_pow, 1450
- mp_rpow, 1450
- mp_xtom, 1450
- mutex — concepts relating to mutual exclusion locks, 1472
 - Caveats, 1473
 - Initialization, 1472
- mutex_destroy — mutual exclusion locks, 1475
- mutex_init — mutual exclusion locks, 1475
 - Destroy, 1479
 - Dynamically Allocated Mutexes, 1485
 - Initialize, 1475
 - Interprocess Locking, 1482
 - Lock and Unlock, 1478
 - Multiple Instruction Single Data, 1481
 - Single Gate, 1480
 - Solaris Interprocess Robust Locking, 1484
- mutex_lock — mutual exclusion locks, 1475
- mutex_trylock — mutual exclusion locks, 1475
- mutex_unlock — mutual exclusion locks, 1475
- mutual exclusion locks
 - mutex_destroy, 1475
 - mutex_init, 1475
 - mutex_lock, 1475
 - mutex_trylock, 1475
 - mutex_unlock, 1475
- mvaddch — add a character (with rendition) to a window, 182
- mvaddchnstr — copy a character string (with renditions) to a window, 184
- mvaddchstr — copy a character string (with renditions) to a window, 184
- mvaddnstr — add a multi-byte character string (without rendition) to a window, 186
- mvaddnwstr — add a wide-character string to a window, 188, 373
- mvaddstr — add a multi-byte character string (without rendition) to a window, 186
- mvaddwch — add a wchar_t character (with attributes) to a curses window and advance cursor, 368
- mvaddwchnstr — add string of wchar_t characters (and attributes) to a curses window, 371
- mvaddwchstr — add string of wchar_t characters (and attributes) to a curses window, 371
- mvaddwstr — add a wide-character string to a window, 188, 373
- mvadd_wch — add a complex character (with rendition) to a window, 193
- mvadd_wchnstr — copy a string of complex characters (with renditions) to a window, 195
- mvadd_wchstr — copy a string of complex characters (with renditions) to a window, 195
- mvchgat — change the rendition of characters in a window, 294
- mvcur — move the cursor, 1488
- mvdelch — remove a character, 510
- mvderwin — map area of parent window to subwindow, 1489
- mvgetch — get a single-byte character from terminal, 975
- mvgetnstr — get a multibyte character string from terminal, 1026
- mvgetnwstr — get wchar_t character strings from curses terminal keyboard, 431
- mvgetn_wstr — get a wide character string from terminal, 1028
- mvgetstr — get a multibyte character string from terminal, 1026
- mvgetwch — get (or push back) wchar_t characters from curses terminal keyboard, 426
- mvgetwstr — get wchar_t character strings from curses terminal keyboard, 431
- mvget_wch — get a wide character from terminal, 1103
- mvget_wstr — get a wide character string from terminal, 1028

mvhline — use single-byte characters (and renditions) to draw lines, 1120
mvhline_set — use complex characters (and renditions) to draw lines, 1122
mvinch — return a single-byte character (with rendition), 1141
mvinchnstr — retrieve a single-byte character string (with rendition), 1142
mvinchstr — retrieve a single-byte character string (with rendition), 1142
mvinnstr — retrieve a multibyte character string (without rendition), 1153
mvinnwstr — get a string of `wchar_t` characters from a curses window, 457, 1155
mvinsch — insert a character, 1157
mvinsnstr — insert a multibyte character string, 1160
mvinsnstr — insert `wchar_t` string before character under the cursor in a curses window, 451
mvinsstr — insert a multibyte character string, 1160
mvinstr — retrieve a multibyte character string (without rendition), 1153
mvinswch — insert a `wchar_t` character before the character under the cursor in a curses window, 450
mvinswstr — insert `wchar_t` string before character under the cursor in a curses window, 451
mvins_nwstr — insert a wide character string, 1162
mvins_wch — insert a complex character, 1165
mvins_wstr — insert a wide character string, 1162
mvinwch — get a `wchar_t` character and its attributes from a curses window, 453
mvinwchnstr — get a string of `wchar_t` characters (and attributes) from a curses window, 455
mvinwchstr — get a string of `wchar_t` characters (and attributes) from a curses window, 455
mvnwstr — get a string of `wchar_t` characters from a curses window, 457, 1155
mvn_wch — retrieve a complex character (with rendition), 1167
mvn_wchnstr — retrieve complex character string (with rendition), 1168
mvn_wchstr — retrieve complex character string (with rendition), 1168
mvprintw — write formatted output to window, 1490
mvscanw — read formatted input from window, 1492
mvvline — use single-byte characters (and renditions) to draw lines, 1120
mvvline_set — use complex characters (and renditions) to draw lines, 1122
mvwaddch — add a character (with rendition) to a window, 182
mvwaddchnstr — copy a character string (with renditions) to a window, 184
mvwaddchstr — copy a character string (with renditions) to a window, 184
mvwaddnwstr — add a wide-character string to a window, 188, 373
mvwaddstr — add a multi-byte character string (without rendition) to a window, 186
mvwaddwch — add a `wchar_t` character (with attributes) to a curses window and advance cursor, 368
mvwaddwchnstr — add string of `wchar_t` characters (and attributes) to a curses window, 371
mvwaddwchstr — add string of `wchar_t` characters (and attributes) to a curses window, 371
mvwaddwstr — add a wide-character string to a window, 188, 373
mvwadd_wch — add a complex character (with rendition) to a window, 193
mvwadd_wchnstr — copy a string of complex characters (with renditions) to a window, 195

mvwadd_wchstr — copy a string of complex characters (with renditions) to a window, 195
mvwchgat — change the rendition of characters in a window, 294
mvwdelch — remove a character, 510
mvwgetch — get a single-byte character from terminal, 975
mvwgetnstr — get a multibyte character string from terminal, 1026
mvwgetnwstr — get `wchar_t` character strings from curses terminal keyboard, 431
mvwgetn_wstr — get a wide character string from terminal, 1028
mvwgetstr — get a multibyte character string from terminal, 1026
mvwgetwch — get (or push back) `wchar_t` characters from curses terminal keyboard, 426
mvwgetwstr — get `wchar_t` character strings from curses terminal keyboard, 431
mvwget_wch — get a wide character from terminal, 1103
mvwget_wstr — get a wide character string from terminal, 1028
mvwhline — use single-byte characters (and renditions) to draw lines, 1120
mvwhline_set — use complex characters (and renditions) to draw lines, 1122
mvwin — move window, 1494
mvwinch — return a single-byte character (with rendition), 1141
mvwinchnstr — retrieve a single-byte character string (with rendition), 1142
mvwinchstr — retrieve a single-byte character string (with rendition), 1142
mvwinnstr — retrieve a multibyte character string (without rendition), 1153
mvwinnwstr — get a string of `wchar_t` characters from a curses window, 457, 1155
mvwinsch — insert a character, 1157
mvwinsnstr — insert a multibyte character string, 1160
mvwinsnwstr — insert `wchar_t` string before character under the cursor in a curses window, 451
mvwinsstr — insert a multibyte character string, 1160
mvwinstr — retrieve a multibyte character string (without rendition), 1153
mvwinswch — insert a `wchar_t` character before the character under the cursor in a curses window, 450
mvwinswstr — insert `wchar_t` string before character under the cursor in a curses window, 451
mvwins_nstr — insert a wide character string, 1162
mvwins_nwstr — insert a wide character string, 1162
mvwins_wch — insert a complex character, 1165
mvwinwch — get a `wchar_t` character and its attributes from a curses window, 453
mvwinwchnstr — get a string of `wchar_t` characters (and attributes) from a curses window, 455
mvwinwchstr — get a string of `wchar_t` characters (and attributes) from a curses window, 455
mvwinwstr — get a string of `wchar_t` characters from a curses window, 457, 1155
mvwin_wch — retrieve a complex character (with rendition), 1167
mvwin_wchnstr — retrieve complex character string (with rendition), 1168
mvwin_wchstr — retrieve complex character string (with rendition), 1168
mvwprintw — write formatted output to window, 1490
mvwscanw — read formatted input from window, 1492

mvwvline — use single-byte characters (and renditions) to draw lines, 1120

mvwvline_set — use complex characters (and renditions) to draw lines, 1122

mwwaddnstr — add a multi-byte character string (without rendition) to a window, 186

N

named pipe

create a new one — mkfifo, 1437

nanosleep — high resolution sleep, 1495

napms — sleep process for a specified length of time, 1497

natural logarithm function — log, 1347

netdir — generic transport name-to-address translation, 1498

netdir_free — generic transport name-to-address translation, 1498

netdir_getbyaddr — generic transport name-to-address translation, 1498

netdir_getbyname — generic transport name-to-address translation, 1498

netdir_mergeaddr — generic transport name-to-address translation, 1498

netdir_options — generic transport name-to-address translation, 1498

netdir_perror — generic transport name-to-address translation, 1498

netdir_sperror — generic transport name-to-address translation, 1498

network configuration database entry

— endnetconfig, 1019

— freenetconfig, 1019

— getnetconfig, 1019

— getnetconfig, 1019

— nc_perror, 1019

— nc_sperror, 1019

— setnetconfig, 1019

network configuration entry corresponding to NETPATH

— endnetpath, 1024

— getnetpath, 1024

— setnetpath, 1024

network entry

— endnetent, 1015

— getnetbyaddr, 1015

— getnetbyaddr_r, 1015

— getnetbyname, 1015

— getnetbyname_r, 1015

— getnetent, 1015

— getnetent_r, 1015

— setnetent, 1015

network group entry

— endnetgrent, 1021

— getnetgrent, 1021

— getnetgrent_r, 1021

— innnetgr, 1021

— setnetgrent, 1021

network host entry

— endhostent, 998

— gethostbyaddr, 998

— gethostbyaddr_r, 998

— gethostbyname, 998

— gethostbyname_r, 998

— gethostent, 998

— gethostent_r, 998

— sethostent, 998

network listener service

format and send listener service request message — nlsrequest, 1560

get client's data passed via the listener — nlsgetcall, 1557

get name of transport provider — nlsprovider, 1559

network protocol entry

— endprotoent, 1041

— getprotobyname, 1041

— getprotobyname_r, 1041

— getprotobynumber, 1041

— getprotobynumber_r, 1041

— getprotoent, 1041

— getprotoent_r, 1041

— setprotoent, 1041

newpad — create and display curses
 pads, 468, 1504
 newterm — screen initialization
 functions, 1152
 newwin — create a new window or
 subwindow, 518
 next representable double-precision
 floating-point number —
 nextafter, 1505
 nextafter — next representable
 double-precision
 floating-point number, 1505
 nextkey — data base subroutines, 499
 nftw — walk a file tree, 936
 nice — change priority of a process, 1506
 NIS client interface
 — ypcnt, 2555
 — yperr_string, 2555
 — ypprot_err, 2555
 — yp_all, 2555
 — yp_bind, 2555
 — yp_first, 2555
 — yp_get_default_domain, 2555
 — yp_master, 2555
 — yp_match, 2555
 — yp_next, 2555
 — yp_order, 2555
 — yp_unbind, 2555
 NIS+ table functions — nis_tables
 nis_first_entry, 1543
 nis_modify_entry, 1543
 nis_next_entry, 1543
 nis_remove_entry, 1543
 NIS+ database functions
 — db_add_entry, 1508
 — db_checkpoint, 1508
 — db_create_table, 1508
 — db_destroy_table, 1508
 — db_first_entry, 1508
 — db_free_result, 1508
 — db_initialize, 1508
 — db_list_entries, 1508
 — db_next_entry, 1508
 — db_remove_entry, 1508
 — db_reset_next_entry, 1508
 — db_standby, 1508
 — db_table_exists, 1508
 — db_unload_table, 1508
 — nis_db, 1508
 NIS+ error messages
 nis_error, 1512
 nis_lerror, 1512
 nis_perror, 1512
 nis_sperrno, 1512
 nis_sperro, 1512
 nis_sperro_r, 1512
 NIS+ group manipulation functions
 — nis_addmember, 1514
 — nis_creategroup, 1514
 — nis_destroygroup, 1514
 — nis_groups, 1514
 — nis_ismember, 1514
 — nis_print_group_entry, 1514
 — nis_removemember, 1514
 — nis_verifygroup, 1514
 NIS+ local names
 — nis_freenames, 1540
 — nis_getnames, 1540
 — nis_local_directory, 1517
 — nis_local_group, 1517
 — nis_local_host, 1517
 — nis_local_names, 1517
 — nis_local_principal, 1517
 NIS+ log administration functions
 — nis_checkpoint, 1536
 — nis_ping, 1536
 NIS+ miscellaneous functions
 — nis_freeservelist, 1538
 — nis_freetags, 1538
 — nis_getservlist, 1538
 — nis_mkdir, 1538
 — nis_rmdir, 1538
 — nis_server, 1538
 — nis_servstate, 1538
 — nis_stats, 1538
 NIS+ namespace functions
 — nis_add, 1519
 — nis_freeresult, 1519
 — nis_lookup, 1519
 — nis_modify, 1519
 — nis_names, 1519
 — nis_remove, 1519
 NIS+ object formats
 — nis_objects, 1526
 NIS+ subroutines

- nis_clone_object, 1540
- nis_destroy_object, 1540
- nis_dir_cmp, 1540
- nis_domain_of, 1540
- nis_leaf_of, 1540
- nis_name_of, 1540
- nis_print_object, 1540
- nis_subr, 1540

NIS+ table functions

- nis_add_entry, 1543
- nis_first_entry, 1543
- nis_list, 1543
- nis_modify_entry, 1543
- nis_next_entry, 1543
- nis_remove_entry, 1543
- nis_tables, 1543

NIS, change information

- yp_update, 2561

nis_tables — NIS+ table functions, 1543

nis_tables — NIS+ table functions, 1543

nis_tables — NIS+ table functions, 1543

nl — enable/disable newline control, 1553

nlist — get entries from symbol table, 1554

nl_langinfo — language information, 1556

nocbreak — set input mode controls, 289

nodelay — set blocking or non-blocking read, 1562

noecho — enable/disable terminal echo, 634

non-local goto — setjmp, 2008, 2012

- longjmp, 2008, 2012
- _longjmp, 2008
- _setjmp, 2008
- siglongjmp, 2012
- sigsetjmp, 2012

non-local goto

- _longjmp, 1349
- _setjmp, 1349

nonl — enable/disable newline control, 1553

noqiflush — control flush of input and output on interrupt, 1563

noraw — set input mode controls, 289

NOTE — annotate source code with info for tools, 1564

- NOTE vs _NOTE, 1565
- NoteInfo Argument, 1565

_NOTE — annotate source code with info for tools, 1564

notimeout — set timed blocking or non-blocking read, 1567

numbers, convert to strings — econvert, 637

O

offsetof — offset of structure member, 1568

open a shared object — dlopen, 577

open a stream — fopen, 864, 922

open directory — opendir, 1569

opendir — open directory, 1569

openlog — control system log, 2142

openpl — graphics interface, 1638

openvt — graphics interface, 1638

operations on a synchronization object in libthread_db

- td_sync_get_info, 2184
- td_sync_setstate, 2184
- td_sync_waiters, 2184

output conversion

- wsprintf — convert to wchar_t string, 2520

output conversion, formatted

- fprintf, 1647
- printf, 1647
- sprintf, 1647
- vsprintf, 1647
- vprintf, 1647
- vsprintf, 1647

overlap or overwrite windows

- overlay, 1571
- overwrite, 1571

overlay — overlap or overwrite windows, 1571

overlay or overwrite any portion of window

- copywin, 342

overview of concepts related to POSIX thread cancellation — cancellation, 276

overview of the XFN interface — xfn, 2535

an overview of XFN attribute operations — xfn_attributes, 2536

XFN compound syntax: an overview of XFN model for compound name parsing — xfn_compound_names, 2541

overwrite — overlap or overwrite windows, 1571

P

- p2close — close pipes to and from a command, 1574
- p2open — open pipes to and from a command, 1574
- page size, system
 - get — getpagesize, 1033
- pair_content — manipulate color information, 283
- PAIR_NUMBER — manipulate color information, 283
- PAM — Pluggable Authentication Module, 1576, 1600
 - Administrative Interface, 1578
 - Interface Overview, 1576
 - Stacking Multiple Schemes, 1578
 - Stateful Interface, 1577
- PAM error messages
 - get string — pam_strerror, 1618
- PAM routines to maintain module specific state
 - pam_get_data, 1596
 - pam_set_data, 1596
- PAM Service Module APIs
 - PAM, 1600
- pam_acct_mgmt — perform PAM account validation procedures, 1580
- pam_authenticate — perform authentication within the PAM framework, 1582
- pam_chauthtok — perform password related functions within the PAM framework, 1584
- pam_close_session — perform PAM session creation and termination operations, 1590
- pam_end — authentication transaction routines for PAM, 1615
- pam_getenv — returns the value for a PAM environment name, 1586
- pam_getenvlist — returns a list of all the PAM environment variables, 1587
- pam_get_data — PAM routines to maintain module specific state, 1596
- pam_get_item — authentication information routines for PAM, 1598
- pam_open_session — perform PAM session creation and termination operations, 1590
- pam_putenv — change or add a value to the PAM environment, 1592
- pam_setcred — modify/delete user credentials for an authentication service, 1594
- pam_set_data — PAM routines to maintain module specific state, 1596
- pam_set_item — authentication information routines for PAM, 1598
- pam_sm — PAM Service Module APIs
 - Interaction with the User, 1601
 - Interface Overview, 1600
 - Stateful Interface, 1601
- pam_sm_acct_mgmt — service provider implementation for pam_acct_mgmt, 1604
- pam_sm_authenticate — service provider implementation for pam_authenticate, 1606
- pam_sm_chauthtok — service provider implementation for pam_chauthtok, 1608
- pam_sm_close_session — Service provider implementation for pam_open_session and pam_close_session, 1611
- pam_sm_open_session — Service provider implementation for pam_open_session and pam_close_session, 1611
- pam_sm_setcred — service provider implementation for pam_setcred, 1613
- pam_start — authentication transaction routines for PAM, 1615
- panel library
 - See also* curses library,
- panels — character based panels package, 1622
- panels deck manipulation routines
 - bottom_panel, 1625
 - hide_panel, 1624
 - panel_hidden, 1624
 - panel_show, 1624
 - panel_top, 1625

- show_panel, 1624
- top_panel, 1625
- panels deck traversal primitives
 - panel_above, 1619
 - panel_below, 1619
- panels panel, associate application data
 - panel_userptr, 1627
 - set_panel_userptr, 1627
- panels panel, get or set current window
 - panel_window, 1628
 - replace_panel, 1628
- panels virtual screen refresh routine
 - panel_update, 1626
 - update_panel, 1626
- panels window on virtual screen, move
 - move_panel, 1620
 - panel_move, 1620
- panels, create and destroy
 - del_panel, 1621
 - new_panel, 1621
 - panel_new, 1621
- password databases
 - lock the lock file — lckpddf, 1226
 - unlock the lock file — ulckpddf, 1226
- password encryption functions
 - crypt, 347
- passwords
 - get password entry from a file — fgetpwent, 1047
 - get passwd entry from UID — getpw, 1046
 - write password file entry — putpwent, 1775
- passwords, shadow
 - get shadow password database entry (reentrant) — fgetspent_r, 1075
 - write shadow password file entry — putspent, 1777
- path name
 - return last element — path name, 238
- pathfind — search for named file in named directories, 1629
- pclose — initiate pipe to/from a process, 1642
- pechochar — create and display curses pads, 468, 1631
- pechowchar — create and display curses pads, 468
- pecho_wchar — add character and refresh window, 1631
- perform an LDAP add operation
 - ldap_add, 1239
 - ldap_add_ext, 1239
 - ldap_add_ext_s, 1239
 - ldap_add_s, 1239
- perform authentication within the PAM framework — pam_authenticate, 1582
- perform PAM account validation procedures
 - pam_acct_mgmt, 1580
- perform PAM session creation and termination operations
 - pam_close_session, 1590
 - pam_open_session, 1590
- perform password related functions within the PAM framework — pam_chauthtok, 1584
- perform word expansions
 - wordexp, 2516
 - wordfree, 2516
- pererr — print system error messages, 1632
- pfmt — display error message in standard format, 1633
- pipes
 - initiate to/from a process — pclose, 1642
 - open, close to and from a command — p2open, p2close, 1574
- placeholder for future logging functionality — td_log, 2183
- plock — lock or unlock into memory process, text, or data, 1637
- plot — graphics interface, 1638
 - Link Editor, 1639
- Pluggable Authentication Module
 - PAM, 1576
- pnoutrefresh — create and display curses pads, 468, 1504
- point — graphics interface, 1638
- pop a thread cancellation cleanup handler — pthread_cleanup_pop, 1694
- popen — initiate pipe to/from a process, 1642
- pow — power function, 1644
- power function — pow, 1644
- preemption control
 - schedctl_exit, 1944

- schedctl_init, 1944
- schedctl_lookup, 1944
- schedctl_start, 1944
- schedctl_stop, 1944
- prefresh — create and display cursors
 - pads, 468, 1504
- print formatted output
 - fprintf, 1652
 - printf, 1652
 - snprintf, 1652
 - sprintf, 1652
- print formatted output of a variable argument
 - list
 - vfprintf, 2467
 - vprintf, 2467
 - vsnprintf, 2467
 - vsprintf, 2467
- print formatted wide-character output
 - fwprintf, 940
 - swprintf, 940
 - wprintf, 940
- printf — formatted output conversion, 1647, 1652
- printw — write formatted output to
 - window, 1490
- probe insertion interface
 - TNF_DEBUG, 2337
 - TNF_PROBE_0, 2337
 - TNF_PROBE_0_DEBUG, 2337
 - TNF_PROBE_1, 2337
 - TNF_PROBE_1_DEBUG, 2337
 - TNF_PROBE_2, 2337
 - TNF_PROBE_2_DEBUG, 2337
 - TNF_PROBE_3, 2337
 - TNF_PROBE_3_DEBUG, 2337
 - TNF_PROBE_4, 2337
 - TNF_PROBE_4_DEBUG, 2337
 - TNF_PROBE_5, 2337
 - TNF_PROBE_5_DEBUG, 2337
- Process Code string operations —
 - wstring, 2522
 - wscasecmp, 2522
 - wscol, 2522
 - wsdup, 2522
 - wsncasecmp, 2522
- process and LWP control in libthread_db
 - ps_kill, 1672
 - ps_lcontinue, 1672
 - ps_lrolltoaddr, 1672
 - ps_lstop, 1672
 - ps_pcontinue, 1672
 - ps_pstop, 1672
- process service interfaces — proc_service, 1663
- process statistics
 - prepare execution profile — monitor, 1447
- processes
 - change priority — nice, 1506
 - duplicate an open file descriptor — dup2, 632
 - generate path name for controlling terminal — ctermid, ctermid_r, 351
 - get character-string representation — cuserid, 497
 - initiate pipe to/from a process — popen, pclose, 1642
 - manipulate user contexts — makecontext, swapcontext, 1357
 - memory lock or unlock — plock, 1637
 - prepare execution profile — monitor, 1447
 - report CPU time used — clock, 304
 - send signal to a process group — killpg, 1196
 - send signal to program — raise, 1781
 - set terminal foreground process group id — tcsetpgrp, 2181
 - suspend execution for interval — sleep, 2059
 - terminate process — exit, 710
 - terminate the process abnormally — abort, 165
 - wait for process to terminate or stop — WIFSTOPPED, 2475
- proc_service — process service interfaces, 1663
 - SPARC, 1662
 - x86, 1662
- profiling utilities
 - prepare process execution profile — monitor, 1447
- program assertion
 - verify — assert, 220
- program messages
 - open/close a message catalog — catopen, catclose, 286
 - read — catgets, 285

programs
 last locations — end, etext, edata, 694
 provide a transient program number
 — reg_ci_callback, 1815
 pseudo-terminal device
 get name of the slave pseudo-terminal
 device — ptsname, 1771
 grant access to the slave pseudo-terminal
 device — grantpt, 1117
 pseudorandom number functions
 — initstate, 1784
 — random, 1784
 — setstate, 1784
 — srandom, 1784
 psiginfo — system signal messages, 1667
 psignal — system signal messages, 1666, 1667
 ps_kill — process and LWP control in
 libthread_db, 1672
 ps_lcontinue — process and LWP control in
 libthread_db, 1672
 ps_lgetfpregs — routines that access the target
 process register in
 libthread_db, 1668
 ps_lgetregs — routines that access the target
 process register in
 libthread_db, 1668
 ps_lgetxregs — routines that access the target
 process register in
 libthread_db, 1668
 ps_lgetxregsize — routines that access the
 target process register in
 libthread_db, 1668
 ps_lrolltoaddr — process and LWP control in
 libthread_db, 1672
 ps_lsetfpregs — routines that access the target
 process register in
 libthread_db, 1668
 ps_lsetregs — routines that access the target
 process register in
 libthread_db, 1668
 ps_lsetxregs — routines that access the target
 process register in
 libthread_db, 1668
 ps_lstop — process and LWP control in
 libthread_db, 1672
 ps_pcontinue — process and LWP control in
 libthread_db, 1672
 ps_phread — interfaces in libthread_db that
 target process memory
 access, 1671
 ps_pdwrite — interfaces in libthread_db that
 target process memory
 access, 1671
 ps_pglobal_lookup — look up a symbol in the
 symbol table of the load
 object in the target
 process, 1670
 ps_pglobal_sym — look up a symbol in the
 symbol table of the load
 object in the target
 process, 1670
 ps_pstop — process and LWP control in
 libthread_db, 1672
 ps_ptread — interfaces in libthread_db that
 target process memory
 access, 1671
 ps_ptwrite — interfaces in libthread_db that
 target process memory
 access, 1671
 pthread_atfork — register fork handlers, 1674
 pthread_attr_destroy — initialize and destroy
 threads attribute object, 1690
 pthread_attr_getdetachstate — get or set
 detachstate attribute, 1676
 pthread_attr_getguardsize — get or set the
 thread guardsize
 attribute, 1678
 pthread_attr_getinheritsched — get or set
 inheritsched attribute, 1680
 pthread_attr_getschedparam — get or set
 schedparam attribute, 1682
 pthread_attr_getschedpolicy — get or set
 schedpolicy attribute, 1684
 pthread_attr_getscope — get or set
 contentionscope
 attribute, 1686
 pthread_attr_getstackaddr — get or set
 stackaddr attribute, 1688
 pthread_attr_getstacksize — get or set
 stacksize attribute, 1689
 pthread_attr_init — initialize and destroy
 threads attribute object, 1690
 pthread_attr_setdetachstate — get or set
 detachstate attribute, 1676

`pthread_attr_setguardsize` — get or set the thread guardsize attribute, 1678
`pthread_attr_setinheritsched` — get or set inheritsched attribute, 1680
`pthread_attr_setschedparam` — get or set schedparam attribute, 1682
`pthread_attr_setschedpolicy` — get or set schedpolicy attribute, 1684
`pthread_attr_setscope` — get or set contentscope attribute, 1686
`pthread_attr_setstackaddr` — get or set stackaddr attribute, 1688
`pthread_attr_setstacksize` — get or set stacksize attribute, 1689
`pthread_cleanup_pop` — pop a thread cancellation cleanup handler, 1694
`pthread_cleanup_push` — push a thread cancellation cleanup handler, 1695
`pthread_condattr_destroy` — initialize or destroy condition variable attributes object, 1698
`pthread_condattr_getpshared` — get or set the process-shared condition variable attributes, 1696
`pthread_condattr_init` — initialize or destroy condition variable attributes object, 1698
`pthread_condattr_setpshared` — get or set the process-shared condition variable attributes, 1696
`pthread_cond_broadcast` — signal or broadcast a condition, 1702
`pthread_cond_destroy` — initialize or destroy condition variables, 1700
`pthread_cond_init` — initialize or destroy condition variables, 1700
`pthread_cond_signal` — signal or broadcast a condition, 1702
`pthread_cond_timedwait` — wait on a condition, 1704
`pthread_cond_wait` — wait on a condition, 1704
`pthread_create` — create a thread, 1707
`pthread_detach` — detach a thread, 1711
`pthread_equal` — compare thread IDs, 1712
`pthread_exit` — terminate calling thread, 1713
`pthread_getconcurrency` — get or set level of concurrency, 1715
`pthread_getschedparam` — access dynamic thread scheduling parameters, 1717
`pthread_getspecific` — manage thread-specific data, 1719
`pthread_join` — wait for thread termination, 1721
`pthread_key_create` — create thread-specific data key, 1723
`pthread_key_delete` — delete thread-specific data key, 1725
`pthread_mutexattr_destroy` — initialize and destroy mutex attributes object, 1737
`pthread_mutexattr_getprioceiling` — get and set prioceiling attribute of mutex attribute object, 1727
`pthread_mutexattr_getprotocol` — get and set protocol attribute of mutex attribute object, 1729
`pthread_mutexattr_getpshared` — get and set process-shared attribute, 1732
`pthread_mutexattr_gettype` — get or set a mutex type, 1734
`pthread_mutexattr_init` — initialize and destroy mutex attributes object, 1737
`pthread_mutexattr_setprioceiling` — get and set prioceiling attribute of mutex attribute object, 1727
`pthread_mutexattr_setprotocol` — get and set protocol attribute of mutex attribute object, 1729
`pthread_mutexattr_setpshared` — get and set process-shared attribute, 1732
`pthread_mutexattr_settype` — get or set a mutex type, 1734
`pthread_mutex_destroy` — initialize or destroy a mutex, 1741
`pthread_mutex_getprioceiling` — change the priority ceiling of a mutex, 1739

pututline — access utmp file entry, 1093
 pututxline — access utmpx file entry, 1096
 putw — put a byte on a stream, 912
 putwin — read a window from, and write a window to, a file, 1108
 putws — convert a string of Process Code characters to EUC characters and put it on a stream, 1778

Q

qconvert — convert number to ASCII, 637
 qfconvert — convert number to ASCII, 637
 qqconvert — convert number to ASCII, 637
 qiflush — control flush of input and output on interrupt, 1563
 qsort — quick sort, 1779
 quadruple_to_decimal — decimal record from quadruple-precision floating, 736
 queues
 insert/remove element from a queue — insque, remque, 1164

R

rac_drop() — remote asynchronous calls, 1889
 rac_poll() — remote asynchronous calls, 1889
 rac_recv() — remote asynchronous calls, 1889
 rac_send() — remote asynchronous calls, 1889
 radix-independent exponent — logb, 1348
 raise — send signal to program, 1781
 rand — simple random number generator, 1782, 1783
 random — pseudorandom number functions, 1784
 random number generator
 — drand48, 629
 — erand48, 629
 — jrand48, 629
 — lcong48, 629
 — lrand48, 629
 — mrand48, 629
 — nrand48, 629
 — rand, 1782
 — seed48, 629
 — srand48, 629
 random number generator, simple

— rand, 1783
 — srand, 1783
 raw — set input mode controls, 289
 rcmd — execute command remotely, 1787
 read a string of characters without echo — getpass, 1034, 1794
 — kvm_getproc, 1218
 — kvm_nextproc, 1218
 — kvm_setproc, 1218
 write_vtoc, 1794
 read a directory entry — readdir, 1789
 read a string of characters without echo — getpassphrase, 1034
 read a window from, and write a window to, a file
 — getwin, 1108
 — putwin, 1108
 read directory
 — readdir, 1791
 — readdir_r, 1791
 read formatted input from window
 — mvscanw, 1492
 — mvwscanw, 1492
 — scanw, 1492
 — vwscanw, 1492
 — vw_scanw, 1492
 — wscanw, 1492
 read or write asynchronous I/O operations
 — aioread, 204
 — aiowrite, 204
 read or write kstat data
 — kstat_read, 1215
 — kstat_write, 1215
 readdir — read a directory entry, 1789, 1791
 POSIX, 1791
 readdir_r — read directory, 1791
 reading and writing thread registers in libthread_db
 — td_thr_getfpregs, 2205
 — td_thr_getgregs, 2205
 — td_thr_getxregs, 2205
 — td_thr_getxregsize, 2205
 — td_thr_setfpregs, 2205
 — td_thr_setgregs, 2205
 — td_thr_setxregs, 2205
 read_vtoc — read and write a disk's VTOC, 1794

- realloc — memory allocator, 267
- realpath — resolve pathname, 1796
- reboot — reboot system or halt processor, 1798
- receive a message from a socket — recv, 1800
 - recvfrom, 1800
 - recvmsg, 1800
- recv — receive a message from a socket, 1800
- recvfrom — receive a message from a socket, 1800
- recvmsg — receive a message from a socket, 1800
- redraw screen or portion of screen
 - redrawwin, 1814
 - wredrawln, 1814
- redrawwin — redraw screen or portion of screen, 1814
- refresh — refresh windows and lines, 628
- refresh windows and lines
 - doupdate, 628
 - refresh, 628
 - wnoutrefresh, 628
 - wrefresh, 628
- regcomp — compile regular expression, 1816
- regcomp — regular expression matching, 1819
- regerror — regular expression matching, 1819
- regex — execute regular expression, 1816
- regexec — regular expression matching, 1819
- regexpr — regular expression compile and match routines, 1825
- regfree — regular expression matching, 1819
- register callbacks for probe creation and destruction —
 - tnftcl_register_funcs, 2325
- register fork handlers — pthread_atfork, 1674
- regular expression compile and match routines
 - advance, 1825
 - compile, 1825
 - regexpr, 1825
 - step, 1825
- regular expression matching
 - regcomp, 1819
 - regerror, 1819
 - regexec, 1819
 - regfree, 1819
- regular expressions
 - compile and execute — regcomp, regex, 1816
- release removable media device reservation —
 - volmgt_release, 2458
- remainder — remainder function, 1828
- remainder function — remainder, 1828
- remote command, return stream to
 - rcmd, 1787
- remote procedure calls, library routines for —
 - rpc, 1843
- remote system
 - return information about users — rusers, rnusers, 1927
 - write to — rstat, 1925
 - write to — rwall, 1929
- remove — remove file, 1829
- remove a character
 - delch, 510
 - mvdelch, 510
 - mvwdelch, 510
 - wdelch, 510
- remove a line
 - deleteln, 513
 - wdeleteln, 513
- remque — remove element from a queue, 1164
- rename the name of a binding —
 - fn_ctx_rename, 826
- reposition a file-position indicator in a stream
 - fseek, 926
 - fseeko, 926
- reserve removable media device —
 - volmgt_acquire, 2449
- reserve screen line for dedicated purpose —
 - ripoffline, 1842
- reset file position indicator in a stream —
 - rewind, 1837
- reset position of directory stream to the beginning of a directory —
 - rewinddir, 1838
- resetty — restore/save terminal modes, 1830
- reset_prog_mode — save/restore terminal modes, 508
- reset_shell_mode — save/restore terminal modes, 508
- resolve pathname — realpath, 1796
- resolver — resolver routines, 1831
 - dn_comp, 1831
 - dn_expand, 1831
 - res_init, 1831

- res_mkquery, 1831
- res_search, 1831
- res_send, 1831
- resource utilization
 - get information — getrusage, 1056
- restartterm — free space pointed to by terminal, 511
- restore initial terminal environment
 - endwin, 704
 - isendwin, 704
- restore/save terminal modes
 - resetty, 1830
 - savetty, 1830
- res_init — resolver routines, 1831
- res_mkquery — resolver routines, 1831
- res_query — resolver routines, 1831
- res_search — resolver routines, 1831
- res_send — resolver routines, 1831
- retrieve public or secret key —
 - getpublickey, 1045
 - getsecretkey, 1045
 - publickey, 1045
- retrieve a complex character (with rendition)
 - in_wch, 1167
 - mvin_wch, 1167
 - mvwin_wch, 1167
 - win_wch, 1167
- retrieve a multibyte character string (without rendition)
 - innstr, 1153
 - instr, 1153
 - mvinnstr, 1153
 - mvinstr, 1153
 - mvwinnstr, 1153
 - mvwinstr, 1153
 - winnstr, 1153
 - winstr, 1153
- retrieve a single-byte character string (with rendition)
 - inchnstr, 1142
 - inchstr, 1142
 - mvinchnstr, 1142
 - mvinchstr, 1142
 - mvwinchnstr, 1142
 - mvwinchstr, 1142
 - winchnstr, 1142
 - winchstr, 1142
- retrieve a wide character string (without rendition)
 - innwstr, 1155
 - inwstr, 1155
 - mvinnwstr, 1155
 - mvinwstr, 1155
 - mvwinnwstr, 1155
 - mvwinwstr, 1155
 - winnwstr, 1155
 - winwstr, 1155
- retrieve archive symbol table —
 - elf_getarsym, 671
- retrieve class-dependent object file header
 - elf32_getehdr, 642
 - elf32_newehdr, 642
 - elf64_getehdr, 642
 - elf64_newehdr, 642
- retrieve class-dependent program header table
 - elf32_getphdr, 644
 - elf32_newphdr, 644
 - elf64_getphdr, 644
 - elf64_newphdr, 644
- retrieve class-dependent section header
 - elf32_getshdr, 646
 - elf64_getshdr, 646
- retrieve complex character string (with rendition)
 - in_wchnstr, 1168
 - in_wchstr, 1168
 - mvin_wchnstr, 1168
 - mvin_wchstr, 1168
 - mvwin_wchnstr, 1168
 - mvwin_wchstr, 1168
 - win_wchnstr, 1168
 - win_wchstr, 1168
- return physical memory information —
 - system, 2147
 - asystem, 2147
- return a file offset for a file descriptor —
 - tell, 2219
- return a file offset in a stream
 - ftell, 932
 - ftello, 932
- return a
 - handle to the Initial Context —
 - fn_ctx_handle_from_initial, 815

returns a list of all the PAM environment variables —
 pam_getenvlist, 1587

return a single-byte character (with rendition)
 — inch, 1141
 — mvinch, 1141
 — mvwinch, 1141
 — winch, 1141

return character string used as key name
 — keyname, 1194
 — key_name, 1194

return credential information associated with the client — door_cred, 620

return current ERASE or KILL characters
 — erasechar, 705
 — erasewchar, 705
 — killchar, 705
 — killwchar, 705

return from a door invocation —
 door_return, 623

return full terminal type name —
 longname, 1350

return information associated with a door descriptor — door_info, 621

return magnitude of first argument and sign of second argument —
 copysign, 341

return multiple attributes associated with named object
 — fn_attr_multi_get, 778
 — fn_multigetlist_destroy, 778
 — fn_multigetlist_next, 778
 — FN_multigetlist_t, 778

return pathname of executable —
 getexecname, 991

return specified attribute associated with name
 — fn_attr_get, 762

return stream to a remote command —
 rexec, 1839

return terminal baud rate — baudrate, 239

return the size of an object file type
 — elf32_fsize, 641
 — elf64_fsize, 641

return the synchronization handle for the object on which a thread is blocked —
 td_thr_sleepinfo, 2216

returns the value for a PAM environment name — pam_getenv, 1586

return the value of a terminfo capability
 — tigetflag, 2281
 — tigetnum, 2281
 — tigetstr, 2281
 — tparm, 2281

return the value of the environmental variable TERM — termname, 2223

return the video attributes supported by the terminal — termattrs, 2221

return the Volume Management root directory
 — volmgt_root, 2460

return values of an attribute
 — fn_attr_get_values, 766
 — fn_valuelist_destroy, 766
 — fn_valuelist_next, 766
 — FN_valuelist_t, 766

return whether or not Volume Management is running —
 volmgt_running, 2461

returns an unbiased exponent — ilogb, 1139

revoke access to a door descriptor —
 door_revoke, 624

rewind — reset file position indicator in a stream, 1837

rewinddir — reset position of directory stream to the beginning of a directory, 1838

rexec — return stream to a remote command, 1839

re_comp — compile and execute regular expressions, 1799

re_exec — compile and execute regular expressions, 1799

rindex — string operations, 1144

rint — round-to-nearest integral value, 1841

ripcoffline — reserve screen line for dedicated purpose, 1842

rmdirp — remove directories in a path, 1435

rnusers — return information about users on remote machines, 1927

round-to-nearest integral value — rint, 1841

routines that access the target process register in libthread_db
 — ps_lgetfpregs, 1668
 — ps_lgetregs, 1668

- ps_lgetxregs, 1668
- ps_lgetxregsize, 1668
- ps_lsetfregs, 1668
- ps_lsetregs, 1668
- ps_lsetxregs, 1668
- rpc — library routines for remote procedure calls, 1843
- RPC
 - data transmission using XDR routines — xdr, 2524
 - RPC bind service library routines
 - rpcbind, 1853
 - rpcb_getaddr, 1853
 - rpcb_gettime, 1853
 - rpcb_rmtcall, 1853
 - rpcb_set, 1853
 - rpcb_unset, 1853
 - rpc_getmaps, 1853
 - RPC entry
 - endrpcent, 1052
 - getrpcbyname, 1052
 - getrpcbyname_r, 1052
 - getrpcbynumber, 1052
 - getrpcbynumber_r, 1052
 - getrpcent, 1052
 - getrpcent_r, 1052
 - setrpcent, 1052
 - RPC library routine for manipulating global RPC attributes for client and server applications
 - rpc_control, 1868
 - RPC library routines for creation and manipulation of server handles
 - rpc_svc_create, 1915
 - svc_create, 1915
 - svc_destroy, 1915
 - svc_dg_create, 1915
 - svc_fd_create, 1915
 - svc_raw_create, 1915
 - svc_tli_create, 1915
 - svc_tp_create, 1915
 - svc_vc_create, 1915
 - RPC library routines for registering servers
 - rpc_reg, 1921
 - rpc_svc_reg, 1921
 - svc_auth_reg, 1921
 - svc_reg, 1921
 - svc_unreg, 1921
 - xpvt_register, 1921
 - xpvt_unregister, 1921
 - RPC library routines for server side errors
 - rpc_svc_err, 1919
 - svcerr_auth, 1919
 - svcerr_decode, 1919
 - svcerr_noproc, 1919
 - svcerr_noprogram, 1919
 - svcerr_progvers, 1919
 - svcerr_systemerr, 1919
 - svcerr_weakauth, 1919
 - RPC obsolete library routines
 - authdes_create, 1899
 - authunix_create_default, 1899
 - callrpc, 1899
 - clntraw_create, 1899
 - clnttcp_create, 1899
 - clntudp_bufcreate, 1899
 - clntudp_create, 1899
 - clnt_broadcast, 1899
 - getrpcport, 1899
 - get_myaddress, 1899
 - pmap_getmaps, 1899
 - pmap_getport, 1899
 - pmap_rmtcall, 1899
 - pmap_set, 1899
 - pmap_unset, 1899
 - registerrpc, 1899
 - rpc_soc, 1899
 - svcsd_create, 1899
 - svcraw_create, 1899
 - svctcp_create, 1899
 - svcudp_bufcreate, 1899
 - svcudp_create, 1899
 - svc_fds, 1899
 - svc_getcaller, 1899
 - svc_getreq, 1899
 - svc_register, 1899
 - svc_unregister, 1899
 - xdr_authunix_parms, 1899
- rpc routines
 - rac_drop() — remote asynchronous calls, 1889
 - rac_poll() — remote asynchronous calls, 1889

rac_recv() — remote asynchronous calls, 1889
 rac_send() — remote asynchronous calls, 1889
 RPC using Kerberos authentication routines
 — authkerb_getucred, 1191
 — authkerb_seccreate, 1191
 — kerberos_rpc, 1191
 — svc_kerb_reg, 1191
 RPC, secure library routines
 — authdes_getucred, 1961
 — authdes_seccreate, 1961
 — getnetname, 1961
 — host2netname, 1961
 — key_decryptsession, 1961
 — key_encryptsession, 1961
 — key_gendes, 1961
 — key_secretkey_is_set, 1961
 — key_setsecret, 1961
 — netname2host, 1961
 — netname2user, 1961
 — secure_rpc, 1961
 — user2netname, 1961
 RPC, XDR library routines
 — rpc_xdr, 1923
 — xdr_accepted_reply, 1923
 — xdr_authsys_parms, 1923
 — xdr_callhdr, 1923
 — xdr_callmsg, 1923
 — xdr_opaque_auth, 1923
 — xdr_rejected_reply, 1923
 — xdr_replymsg, 1923
 rpc — security flavor incorporating GSS-API onto ONC RPC, 1893
 rpc_broadcast — library routines for client side calls, 1858
 rpc_broadcast_exp — library routines for client side calls, 1858
 rpc_call — library routines for client side calls, 1858
 rpc_clnt_auth — library routines for client side remote procedure call authentication, 1856
 rpc_clnt_calls — library routines for client side calls, 1858
 Routines, 1858
 rpc_clnt_create — library routines for dealing with creation and manipulation of CLIENT handles, 1862
 Routines, 1862
 rpc_createerr — library routines for dealing with creation and manipulation of CLIENT handles, 1862
 rpc_svc_calls — library routines for RPC servers, 1910
 Routines, 1910
 rresvport — get privileged socket, 1787
 rstat — get performance data from remote kernel, 1925
 ruserok — authenticate user, 1787
 rusers — return information about users on remote machines, 1927
 xdr_utmpidlearr, 1927
 rwall — write to specified remote machines, 1929
 rwlock_destroy() — destroy a readers/writer lock, 1930
 rwlock_init() — initialize a readers/writer lock, 1930
 rw_rdlock() — acquire a read lock, 1930
 rw_tryrdlock() — acquire a read lock, 1930
 rw_trywrlock() — acquire a write lock, 1930
 rw_unlock() — unlock a readers/writer lock, 1930
 rw_wrlock() — acquire a write lock, 1930

S
 save/restore terminal modes
 — def_prog_mode, 508
 — def_shell_mode, 508
 — reset_prog_mode, 508
 — reset_shell_mode, 508
 savetty — restore/save terminal modes, 1830
 scalb — load exponent of a radix-independent floating-point number, 1933
 scalbn — load exponent of a radix-independent floating-point number, 1934
 scan a directory
 — alphasort, 1935
 — scandir, 1935
 scandir — scan a directory, 1935

scanf — convert formatted input, 1937
 Conversion Characters, 1939
 Conversion Specifications, 1938
 scanw — read formatted input from
 window, 1492
 schedctl_exit — preemption control, 1944
 schedctl_init — preemption control, 1944
 schedctl_lookup — preemption control, 1944
 schedctl_start — preemption control, 1944
 schedctl_stop — preemption control, 1944
 scheduling priority
 change priority of a process — nice, 1506
 sched_getparam — get scheduling
 parameters, 1946, 1952
 sched_getscheduler — get scheduling
 policy, 1949
 sched_get_priority_max — get scheduling
 parameter limits, 1947
 sched_get_priority_min — get scheduling
 parameter limits, 1947
 sched_rr_get_interval — get execution time
 limits, 1951
 sched_setparam — set/get scheduling
 parameters, 1952
 sched_setscheduler — set scheduling policy
 and scheduling
 parameters, 1955
 sched_yield — yield processor, 1958
 screen initialization functions
 — initscr, 1152
 — newterm, 1152
 scl — scroll a window, 1960
 scroll — scroll a window, 1960
 scroll a window
 — scl, 1960
 — scroll, 1960
 — wscrl, 1960
 scrollok — set terminal output controls, 302
 scr_dump — write screen contents to/from a
 file, 1959
 scr_init — write screen contents to/from a
 file, 1959
 scr_restore — write screen contents to/from a
 file, 1959
 scr_set — write screen contents to/from a
 file, 1959
 search for atomic name with specified
 attributes in single context
 — fn_attr_search, 784
 — fn_searchlist_destroy, 784
 — fn_searchlist_next, 784
 — FN_searchlist_t, 784
 search for names whose attributes satisfy filter
 — fn_attr_ext_search, 755
 — fn_ext_searchlist_destroy, 755
 — fn_ext_searchlist_next, 755
 — FN_ext_searchlist_t, 755
 search functions
 binary search a sorted table —
 bsearch, 270
 linear search and update routine —
 lsearch, lfind, 1351
 manage hash search tables —
 hsearch, 1124
 seconvert — convert number to ASCII, 637
 secure, RPC
 See RPC,secure, 1961
 seekdir — set position of directory
 stream, 1965
 select — synchronous I/O multiplexing, 1966
 semaphore
 acquire or wait for — sem_wait,
 sem_trywait, 1986
 close a named one — sem_close, 1974
 destroy an unnamed one —
 sem_destroy, 1975
 get the value — sem_getvalue, 1976
 increment the count — sem_post, 1982
 initialize an unnamed one —
 sem_init, 1977
 initialize/open a named one —
 sem_open, 1979
 remove a named one — sem_unlink, 1984
 sema_destroy() — destroy a semaphore, 1970
 sema_init() — initialize a semaphore, 1970
 sema_post() — increment a semaphore, 1970
 sema_trywait() — decrement a
 semaphore, 1970
 sema_wait() — decrement a semaphore, 1970
 sem_close — close a named semaphore, 1974
 sem_destroy — destroy an unnamed
 semaphore, 1975
 sem_getvalue — get the value of a
 semaphore, 1976

- sem_init — initialize an unnamed semaphore, 1977
- sem_open — initialize/open a named semaphore, 1979
- sem_post — increment the count of a semaphore, 1982
- sem_trywait — acquire or wait for a semaphore, 1986
- sem_unlink — remove a named semaphore, 1984
- sem_wait — acquire or wait for a semaphore, 1986
- send — send message from a socket, 1989
- send a “break” for a specific duration — tcsendbreak, 2177
- sendmsg — send message from a socket, 1989
- sendto — send message from a socket, 1989
- Service Access Facility library function
 - doconfig, 609
- Service Provider functions for components
 - DmiOriginateEvent, 606
 - DmiRegisterCi, 606
 - DmiUnRegisterCi, 606
- service provider implementation for
 - pam_acct_mgmt —
 - pam_sm_acct_mgmt, 1604
- service provider implementation for
 - pam_authenticate —
 - pam_sm_authenticate, 1606
- service provider implementation for
 - pam_chauthtok —
 - pam_sm_chauthtok, 1608
- Service provider implementation for
 - pam_open_session and
 - pam_close_session
 - pam_sm_close_session, 1611
 - pam_sm_open_session, 1611
- service provider implementation for
 - pam_setcred —
 - pam_sm_setcred, 1613
- set a cchar_t type character from a wide character and rendition — setcchar, 2007
- set and/or get alternate signal stack context — sigstack, 2048
- set and/or get signal stack context — sigstack, 2046
- set blocking or non-blocking read —
 - nodelay, 1562
- set concurrency level for target process —
 - td_ta_setconcurrency, 2200
- set connectionless LDAP request
 - retransmission parameters —
 - cldap_setretryinfo, 300
- set encoding key — setkey, 2015
- set foreground process group ID —
 - tcsetpgrp, 2180
- set input baud rate
 - cfsetispeed, 293
- set input mode controls
 - cbreak, 289
 - nocbreak, 289
 - noraw, 289
 - raw, 289
- set or get the background character (and rendition) of window using a complex character
 - bkgrnd, 259
 - bkgrndset, 259
 - getbkgrnd, 259
 - wbkgrnd, 259
 - wbkgrndset, 259
 - wgetbkgrnd, 259
- set output baud rate
 - cfsetospeed, 293
- set position of directory stream —
 - seekdir, 1965
- set scheduling policy and scheduling parameters —
 - sched_setscheduler, 1955
- set server principal name
 - rpc_gss_set_svc_name, 1887
- set stream orientation — fwide, 939
- set terminal output controls
 - clearok, 302
 - idlok, 302
 - leaveok, 302
 - scrollok, 302
 - setscreg, 302
 - wssetscreg, 302
- set the background character (and rendition) of window
 - bkgd, 257
 - bkgdset, 257

- wbgkd, 257
- wbgkdset, 257
- set the cancellation type of a thread —
pthread_setcanceltype, 1762
- set the parameters associated with the
terminal — tcsetattr, 2178
- set the priority of a thread —
td_thr_setprio, 2213
- set timed blocking or non-blocking read
 - notimeout, 1567
 - timeout, 1567
 - wtimeout, 1567
- set values of lines and columns —
use_env, 2443
- set visibility of cursor — curs_set, 479
- set wide-characters in memory —
wmemset, 2515
- set/clear window attributes
 - standend, 2086
 - standout, 2086
 - wstandend, 2086
 - wstandout, 2086
- set/get scheduling parameters
 - sched_getparam, 1952
 - sched_setparam, 1952
- setac — get audit control file information, 960
- setauclass — rewind audit_class database
file, 962
- setauuser — rewind audit_event database
file, 967
- setauuser — get audit_user database entry, 970
- setcat — define default catalog, 2006
- setcchar — set a cchar_t type character from a
wide character and
rendition, 2007
- setgrent — get group entry from database, 994
- sethostname — set name of current host, 1005
- setjmp — non-local goto, 2008, 2012
- _setjmp — non-local goto, 1349, 2008
- setkey — set encoding key, 2015
- setlabel — define the label for pfmt() and
lfmt()., 2016
- setlocale — modify and query a program's
locale, 2017
- setlogmask — control system log, 2142
- setpriority — get or set process scheduling
priority, 1039
- setpwnam — get password entry from user
database, 1047
- setscreg — set terminal output controls, 302
- setservent — get service entry, 1060
- setspent — get shadow password database
entry, 1075
- setstate — pseudorandom number
functions, 1784
- setterm — free space pointed to by
terminal, 511
- settimeofday — set system's notion of current
Greenwich time, 1086, 1088
- setupterm — free space pointed to by
terminal, 511
- setusershell() — function, 1092
- setutent — access utmp file entry, 1093
- setutxent — access utmpx file entry, 1096
- set_curterm — free space pointed to by
terminal, 511
- set_term — switch between terminals, 2023
- severity levels, applications
 - build a list for use with fmtmsg —
addseverity, 191
- sfconvert — convert number to ASCII, 637
- sgconvert — convert number to ASCII, 637
- shared memory object
 - open — shm_open, 2024
 - remove — shm_unlink, 2027
- shared object
 - close — dlclose, 566
 - get address of symbol — dlsym, 582
 - get diagnostic information — dlerror, 574
 - translate address to symbolic information
— dladdr, 564, 577
- shell command
 - issue one — system, 2148
- shell global pattern matching — gmatch, 1116
- shm_open — open a shared memory
object, 2024
- shm_unlink — remove a shared memory
object, 2027
- shutdown — shut down part of a full-duplex
connection, 2028
- sig2str — translation between signal name and
signal number, 2092
- sigaddset — manipulate sets of signals, 2044
- sigdelset — manipulate sets of signals, 2044

sigemptyset — manipulate sets of signals, 2044
 sigfillset — manipulate sets of signals, 2044
 sigfp() function, 2033
 sighold — adds sig to the calling process's
 signal mask, 2039
 sigignore — sets the disposition of sig to
 SIG_IGN, 2039
 siginterrupt — allow signals to interrupt
 functions, 2036
 sigismember — manipulate sets of signals, 2044
 siglongjmp — non-local goto, 2012
 signal — simplified software signal
 facilities, 2037, 2039
 queue one to a process — sigqueue, 2042
 schedule after interval in microseconds —
 ualarm, 2437
 suspend execution for interval in
 microseconds — usleep, 2444
 wait for queued signals — sigwaitinfo,
 sigtimedwait, 2055
 simplified signal facilities — bsd_signal, 269
 signal management
 simplified, for application processes —
 signal, 2039
 signal messages, system
 — psignal, 1666, 1667
 signal or broadcast a condition
 — pthread_cond_broadcast, 1702
 — pthread_cond_signal, 1702
 signals, block
 — sigblock, 2031
 — sigmask, 2031
 — sigpause, 2031
 — sigsetmask, 2031
 signals, software
 — gsignal, 2085
 — ssignal, 2085
 significand — significand function, 2041
 significand function — significand, 2041
 sigpause — removes sig from the calling
 process's signal mask and
 suspends the calling process
 until a signal is received, 2039
 sigqueue — queue a signal to a process, 2042
 sigrelse — removes sig from the calling
 process's signal mask, 2039
 sigset — modify signal disposition, 2039
 sigsetjmp — non-local goto, 2012
 sigsetops — manipulate sets of signals, 2044
 sigstack — set and/or get signal stack
 context, 2046, 2048
 sigtimedwait — wait for queued signals, 2055
 sigvec — software signal facilities, 2050
 sigwaitinfo — wait for queued signals, 2055
 simplified Basic Encoding Rules library
 encoding functions
 — ber_alloc, 247
 — ber_encode, 247
 — ber_flush, 247
 — ber_printf, 247
 — ber_put_bitstring, 247
 — ber_put_boolean, 247
 — ber_put_int, 247
 — ber_put_null, 247
 — ber_put_ostring, 247
 — ber_put_seq, 247
 — ber_put_set, 247
 — ber_put_string, 247
 — ber_start_seq, 247
 — ber_start_set, 247
 sin — sine function, 2057
 sine function — sin, 2057
 single-byte to wide-character conversion —
 btowc, 273
 single_to_decimal — decimal record from
 single-precision floating, 736
 sinh — hyperbolic sine function, 2058
 sleep — suspend execution for interval, 2059
 high resolution — nanosleep, 1495
 suspend execution for interval in
 microseconds — usleep, 2444
 sleep process for a specified length of time —
 napms, 1497
 slk_attoff — manipulate soft labels, 2062
 slk_attron — manipulate soft labels, 2062
 slk_attrset — manipulate soft labels, 2062
 slk_attr_off — manipulate soft labels, 2062
 slk_attr_on — manipulate soft labels, 2062
 slk_attr_set — manipulate soft labels, 2062
 slk_clear — manipulate soft labels, 2062
 slk_color — manipulate soft labels, 2062
 slk_init — manipulate soft labels, 2062
 slk_label — manipulate soft labels, 2062
 slk_noutrefresh — manipulate soft labels, 2062
 slk_refresh — manipulate soft labels, 2062

slk_restore — manipulate soft labels, 2062
 slk_set — manipulate soft labels, 2062
 slk_touch — manipulate soft labels, 2062
 slk_wset — manipulate soft labels, 2062
 snprintf — print formatted output, 1652
 socket — create an endpoint for
 communication, 2064
 accept a connection — accept, 167
 bind a name — bind, 252
 get options — getsockopt, 1068
 get name — getsockname, 1065
 get name of connected peer —
 getpeername, 1036
 initiate a connection — connect, 332
 listen for connections — listen, 1333
 send message from — send, sendto,
 sendmsg, 1989
 set options — setsockopt, 1068
 shut down part of a full-duplex connection
 — shutdown, 2028
 socketpair — create a pair of connected
 sockets, 2070
 software signals
 — gsignal, 2085
 — ssignal, 2085
 sort
 quick — qsort, 1779
 space — graphics interface, 1638
 specify an alternative door server thread
 creation function —
 door_server_create, 625
 sprintf — formatted output conversion, 1647
 spray — scatter data in order to test the
 network, 2074
 sprintf — print formatted output, 1652
 sqrt — square root function, 2076
 square root function — sqrt, 2076
 srand — reset simple random number
 generator, 1782
 srandom — pseudorandom number
 functions, 1784
 SSAAgentIsAlive — Sun Solstice Enterprise
 Agent registration and
 communication helper
 functions, 2077
 SSAGetTrapPort — Sun Solstice Enterprise
 Agent registration and
 communication helper
 functions, 2077
 SSAOidCmp — Sun Solstice Enterprise Agent
 OID helper functions, 2081
 SSAOidCpy — Sun Solstice Enterprise Agent
 OID helper functions, 2081
 SSAOidDup — Sun Solstice Enterprise Agent
 OID helper functions, 2081
 SSAOidFree — Sun Solstice Enterprise Agent
 OID helper functions, 2081
 SSAOidInit — Sun Solstice Enterprise Agent
 OID helper functions, 2081
 SSAOidNew — Sun Solstice Enterprise Agent
 OID helper functions, 2081
 SSAOidString — Sun Solstice Enterprise Agent
 OID helper functions, 2081
 SSAOidStrToOid — Sun Solstice Enterprise
 Agent OID helper
 functions, 2081
 SSAOidZero — Sun Solstice Enterprise Agent
 OID helper functions, 2081
 SSAREgSubagent — Sun Solstice Enterprise
 Agent registration and
 communication helper
 functions, 2077
 SSAREgSubtable — Sun Solstice Enterprise
 Agent registration and
 communication helper
 functions, 2077
 SSAREgSubtree — Sun Solstice Enterprise
 Agent registration and
 communication helper
 functions, 2077
 SSASendTrap — Sun Solstice Enterprise Agent
 registration and
 communication helper
 functions, 2077
 SSAStrnCpy — Sun Solstice Enterprise Agent
 string helper functions, 2083
 SSAStrnInit — Sun Solstice Enterprise Agent
 string helper functions, 2083
 SSAStrnToChar — Sun Solstice Enterprise
 Agent string helper
 functions, 2083
 SSAStrnZero — Sun Solstice Enterprise
 Agent string helper
 functions, 2083

SSASubagentOpen — Sun Solstice Enterprise Agent registration and communication helper functions, 2077

sscanf — convert formatted input, 1937

standend — curses character and window attribute control routines, 376, 2086

standout — curses character and window attribute control routines, 376, 2086

start_color — manipulate color information, 283

stdio — standard buffered input/output package, 2087

step — regular expression compile and match routines, 1825

step through LDAP entry attributes — ldap_first_attribute, 1268 — ldap_next_attribute, 1268

string collation — strcoll, 2095

store — data base subroutines, 499

strfind — string manipulations, 2097

str2sig — translation between signal name and signal number, 2092

strcadd — copy strings, compressing or expanding C language escape codes, 2093

strcasecmp — string operations, 2108

strcat — string operations, 2108

strccpy — copy strings, compressing or expanding C language escape codes, 2093

strchr — string operations, 2108

strcmp — string operations, 2108

strcpy — string operations, 2108

strcspn — string operations, 2108

strdup — string operations, 2108

streadd — copy strings, compressing or expanding C language escape codes, 2093

stream

- convert a string of EUC characters from the stream to Process Code — getwfs, 1109
- convert a string of Process Code characters to EUC characters and put it on a stream — putwfs, 1778
- open — fopen, 862

stream status inquiries

- clearerr, 725
- feof, 725
- ferror, 725
- fileno, 725

stream, assign buffering

- setbuf, 2002
- setvbuf, 2002

stream, get string

- fgets, 1059
- gets, 1059

stream, put a string

- fputs, 1776
- puts, 1776

STREAMS

- accept a connection on a socket — accept, 167
- attach a STREAMS-based file descriptor to an object in the file system name space — fattach, 714
- bind a name to a socket — bind, 252
- buffered binary input/output — fread, 920
- create a pair of connected sockets — socketpair, 2070
- create an endpoint for communication — socket, 2064
- determine whether a buffer of characters is encrypted — isencrypt, 1174
- get and set socket options — getsockopt, setsockopt, 1068
- get name of peer connected to socket — getpeername, 1036
- get socket name — getsockname, 1065
- initiate a connection on a socket — connect, 332
- listen for connections on a socket — listen, 1333
- read stream up to next delimiter — bgets, 251
- return to remote command — rcmd, 1787
- send a message from a socket — send, sendto, sendmsg, 1989

- shut down part of a full-duplex connection
 - shutdown, 2028
 - split buffer into fields — bufsplit, 274
 - test file descriptor for a STREAMS file —
 - isastream, 1172
- strecpy — copy strings, compressing or expanding C language escape codes, 2093
- strfind — string manipulations, 2097
- strfmon — convert monetary value to string, 2098
- strftime — convert date and time to string, 2103
- string manipulations — strfind, 2097, 2108
 - strrspn, 2097
 - strtrns, 2097
- string conversion routines
 - atoi, 2122
 - atol, 2122
 - atoll, 2122
 - lltostr, 2122
 - strtol, 2122
 - strtoll, 2122
 - ulltostr, 2122
- string encoding function — crypt, 346
- string manipulations
 - strfind, 2097
 - strrspn, 2097
 - strtrns, 2097
- string operation
 - get error message string — sterror, 2096
 - get PAM error message string —
 - pam_sterror, 1618
- string operations
 - bit and byte — bstring, 272
 - index, 1144
 - rindex, 1144
 - strcasecmp, 2108
 - strcat, 2108
 - strchr, 2108
 - strcmp, 2108
 - strcpy, 2108
 - strcspn, 2108
 - strdup, 2108
 - string, 2108
 - strlen, 2108
 - strncasecmp, 2108
 - strncat, 2108
 - strncmp, 2108
 - strncpy, 2108
 - strpbrk, 2108
 - strchr, 2108
 - strspn, 2108
 - strstr, 2108
 - strtok, 2108
 - strtok_r, 2108
- string transformation — strxfrm, 2128
- strings
 - copy, compressing or expanding C language escape codes, 2093
- strings, convert from numbers — econvert, 637
- string_to_decimal — decimal record from character string, 2112
- strlen — string operations, 2108
- strncasecmp — string operations, 2108
- strncat — string operations, 2108
- strncmp — string operations, 2108
- strncpy — string operations, 2108
- strpbrk — string operations, 2108
- strptime — date and time conversion, 2115
- strchr — string operations, 2108
- strfind — string manipulations, 2097
- strsignal — get name of signal, 2119
- strspn — string operations, 2108
- strstr — string operations, 2108
- strtod — convert string to double-precision number, 2120
- strtok — string operations, 2108
- strtok_r — string operations, 2108
- strtol — string conversion routines, 2122
- strtoll — string conversion routines, 2122
- strtoul — convert string to unsigned long, 2125
- strtows — code conversion for Process Code and File Code, 2127
- strfind — string manipulations, 2097
- strxfrm — string transformation, 2128
- subpad — create and display curses pads, 468, 1504
- subwin — create a new window or subwindow, 518
- Sun Solstice Enterprise Agent OID helper functions
 - SSAOidCmp, 2081
 - SSAOidCpy, 2081

- SSAOidDup, 2081
- SSAOidFree, 2081
- SSAOidInit, 2081
- SSAOidNew, 2081
- SSAOidString, 2081
- SSAOidStrToOid, 2081
- SSAOidZero, 2081
- Sun Solstice Enterprise Agent registration and communication helper functions
 - SSAAgentIsAlive, 2077
 - SSAGetTrapPort, 2077
 - SSARegSubagent, 2077
 - SSARegSubtable, 2077
 - SSARegSubtree, 2077
 - SSASendTrap, 2077
 - SSASubagentOpen, 2077
- Sun Solstice Enterprise Agent string helper functions
 - SSAStrCopy, 2083
 - SSAStrInit, 2083
 - SSAStrToChar, 2083
 - SSAStrZero, 2083
- suspend and resume threads in libthread_db
 - td_thr_dbresume, 2203
 - td_thr_dbsuspend, 2203
- suspend or restart the transmission or reception of data — tcflow, 2165
- svc_dg_enablecache — library routines for RPC servers, 1910
- svc_done — library routines for RPC servers, 1910
- svc_exit — library routines for RPC servers, 1910
- svc_fdset — library routines for RPC servers, 1910
- svc_freeargs — library routines for RPC servers, 1910
- svc_getargs — library routines for RPC servers, 1910
- svc_getreqset — library routines for RPC servers, 1910
- svc_getreq_common — library routines for RPC servers, 1910
- svc_getreq_poll — library routines for RPC servers, 1910
- svc_getrpcaller — library routines for RPC servers, 1910
- svc_max_pollfd — library routines for RPC servers, 1910
- svc_pollfd — library routines for RPC servers, 1910
- svc_run — library routines for RPC servers, 1910
- svc_sendreply — library routines for RPC servers, 1910
- swab — swap bytes, 2130
- swap bytes — swab, 2130
- swapcontext — manipulate user contexts, 1357
- switch between terminals — set_term, 2023
- swprintf — print formatted wide-character output, 940
- swscanf — convert formatted wide-character input, 947
- symbol address
 - get address in shared object — dlsym, 582
- symbol table
 - get entries — nlist, 1554
- synchronize a file's data
 - fdatsync, 720
- synchronize window with its parents or children
 - syncok, 2132
 - wcursyncup, 2132
 - wsyncdown, 2132
 - wsyncup, 2132
- synchronous I/O multiplexing
 - FD_CLR, 1966
 - FD_ISSET, 1966
 - FD_SET, 1966
 - select, 1966
- syncok — synchronize window with its parents or children, 2132
- sync_instruction_memory — make modified instructions executable, 2131
- syscall — indirect system call, 2133
- sysconf — get configurable system variables, 2134
- syslog — control system log, 2142
- sysmem — return physical memory information, 2147
- system — issue shell command, 2148
- system error messages

print — perror, 1632
system log
 log message with variable argument list
 — vsyslog, 2469
system signal messages
 — psignal, 1667
system variables
 get configurable ones — sysconf, 2134
sys_siglist — system signal messages list, 1666

T

taddr2uaddr — generic transport
 name-to-address
 translation, 1498
tan — tangent function, 2157
tangent function — tan, 2157
tanh — hyperbolic tangent function, 2158
tcdrain — wait for transmission of
 output, 2164
tcflow — suspend or restart the transmission
 or reception of data, 2165
tcflush — flush non-transmitted output data,
 non-read input data or
 both, 2167
tcgetattr — get the parameters associated with
 the terminal, 2168
tcgetpgrp — get foreground process group
 ID, 2169
tcgetsid — get process group ID for session
 leader for controlling
 terminal, 2170
tcsendbreak — send a “break” for a specific
 duration, 2177
tcsetattr — set the parameters associated with
 the terminal, 2178
tcsetpgrp — set foreground process group
 ID, 2180
tda_ta_clear_event — thread events in
 libthread_db, 2189
tdelete — manage binary search trees, 2398
td_eventisempty — thread events in
 libthread_db, 2189
td_eventismember — thread events in
 libthread_db, 2189
td_event_addset — thread events in
 libthread_db, 2189
td_event_delset — thread events in
 libthread_db, 2189
td_event_emptyset — thread events in
 libthread_db, 2189
td_event_fillset — thread events in
 libthread_db, 2189
td_init — initialization function for
 libthread_db library of
 interfaces, 2182
td_log — placeholder for future logging
 functionality, 2183
td_sync_get_info — operations on a
 synchronization object in
 libthread_db, 2184
td_sync_setstate — operations on a
 synchronization object in
 libthread_db, 2184
td_sync_waiters — operations on a
 synchronization object in
 libthread_db, 2184
td_ta_delete — allocate and deallocate process
 handles for
 libthread_db, 2198
td_ta_enable_stats — collect target process
 statistics for
 libthread_db, 2187
td_ta_event_addr — thread events in
 libthread_db, 2189
 Event Set Manipulation Macros, 2192
td_ta_event_getmsg — thread events in
 libthread_db, 2189
td_ta_get_nthreads — gets the total number of
 threads in a process for
 libthread_db, 2194
td_ta_get_ph — allocate and deallocate
 process handles for
 libthread_db, 2198
td_ta_get_stats — collect target process
 statistics for
 libthread_db, 2187
td_ta_map_addr2sync — get a synchronization
 object handle from a
 synchronization object’s
 address, 2195
td_ta_map_addr2thr — convert a thread id or
 thread address to a thread
 handle, 2196

td_ta_map_id2thr — convert a thread id or thread address to a thread handle, 2196
 td_ta_new — allocate and deallocate process handles for libthread_db, 2198
 td_ta_reset_stats — collect target process statistics for libthread_db, 2187
 td_ta_setconcurrency — set concurrency level for target process, 2200
 td_ta_set_event — thread events in libthread_db, 2189
 td_ta_sync_iter — iterator functions on process handles from libthread_db library of interfaces, 2201
 td_ta_thr_iter — iterator functions on process handles from libthread_db library of interfaces, 2201
 td_ta_tsd_iter — iterator functions on process handles from libthread_db library of interfaces, 2201
 td_thr_clear_event — thread events in libthread_db, 2189
 td_thr_dbresume — suspend and resume threads in libthread_db, 2203
 td_thr_dbsuspend — suspend and resume threads in libthread_db, 2203
 td_thr_event_enable — thread events in libthread_db, 2189
 td_thr_event_getmsg — thread events in libthread_db, 2189
 td_thr_getfpregs — reading and writing thread registers in libthread_db, 2205
 td_thr_getgregs — reading and writing thread registers in libthread_db, 2205
 Intel x86, 2206
 SPARC, 2206
 td_thr_getxregs — reading and writing thread registers in libthread_db, 2205
 td_thr_getxregsize — reading and writing thread registers in libthread_db, 2205
 td_thr_get_info — get thread information in libthread_db library of interfaces, 2208
 td_thr_lockowner — iterate over the set of locks owned by a thread, 2212
 td_thr_setfpregs — reading and writing thread registers in libthread_db, 2205
 td_thr_setgregs — reading and writing thread registers in libthread_db, 2205
 td_thr_setprio — set the priority of a thread, 2213
 td_thr_setsigpending — manage thread signals for libthread_db, 2214
 td_thr_setxregs — reading and writing thread registers in libthread_db, 2205
 td_thr_set_event — thread events in libthread_db, 2189
 td_thr_sigsetmask — manage thread signals for libthread_db, 2214
 td_thr_sleepinfo — return the synchronization handle for the object on which a thread is blocked, 2216
 td_thr_tsd — get a thread's thread-specific data for libthread_db library of interfaces, 2217
 td_thr_validate — test a thread handle for validity, 2218
 tell — return a file offset for a file descriptor, 2219
 telldir — current location of a named directory stream, 2220
 tempnam — create a name for a temporary file, 2297
 termattrs — return the video attributes supported by the terminal, 2221
 terminal
 find the slot in the utmp file of the current user — ttyslot, 2433
 terminal device, slave pseudo
 get name — ptsname, 1771
 grant access — grantpt, 1117
 terminal ID
 generate path name for controlling terminal — ctermid, ctermid_r, 351
 terminal line

- establish an outgoing connection — dial, 525
- terminals
 - set terminal foreground process group id — tcsetpgrp, 2181
- terminate calling thread — pthread_exit, 1713
- terminate the calling thread — thr_exit, 2255
- termios — general terminal interface, 2222
- termname — return the value of the environmental variable TERM, 2223
- test a thread handle for validity — td_thr_validate, 2218
- test character for specified class — iswctype, 1182
- test for a terminal device — isatty, 1173
- test for NaN — isnan, 1179
- text processing utilities
 - compile and execute regular expressions — regcmp, regex, 1816
 - quick sort — qsort, 1779
- text string
 - gettext, 1090
- textdomain — select domain of messages, 1082
- tfind — manage binary search trees, 2398
- tgetent — emulate the termcap database, 2230
- tgetflag — emulate the termcap database, 2230
- tgetnum — emulate the termcap database, 2230
- tgetstr — emulate the termcap database, 2230
- tgoto — emulate the termcap database, 2230
- thread events in libthread_db
 - tda_ta_clear_event, 2189
 - td_eventisempty, 2189
 - td_eventismember, 2189
 - td_event_addset, 2189
 - td_event_delset, 2189
 - td_event_emptyset, 2189
 - td_event_fillset, 2189
 - td_ta_event_addr, 2189
 - td_ta_event_getmsg, 2189
 - td_ta_set_event, 2189
 - td_thr_clear_event, 2189
 - td_thr_event_enable, 2189
 - td_thr_event_getmsg, 2189
 - td_thr_set_event, 2189
- thread yield to another thread — thr_yield, 2280
- thread-specific-data functions
 - thr_getspecific, 2264
 - thr_keycreate, 2264
 - thr_setspecific, 2264
- thr_continue — continue thread execution, 2278
- thr_create — create a thread, 2240
- thr_exit — terminate the calling thread, 2255
- thr_getconcurrency — get thread concurrency level, 2257
- thr_getprio — access dynamic thread scheduling, 2259
 - Contentionscope, 2259
 - Policy, 2260
 - Priority, 2259
 - Scheduling, 2260
- thr_getspecific — thread-specific-data functions, 2264
- thr_join — wait for thread termination, 2262
- thr_keycreate — thread-specific-data functions, 2264
 - Create Key, 2264
 - Get Value, 2265
 - Set Value, 2264
- thr_main — identifies the calling thread as the main thread or not the main thread, 2268
- thr_self — get calling thread's ID, 2271
- thr_setconcurrency — set thread concurrency level, 2257
- thr_setprio — access dynamic thread scheduling, 2259
- thr_setspecific — thread-specific-data functions, 2264
- thr_sigsetmask — change or examine calling thread's signal mask, 2272
- thr_stksegment — get thread stack bottom and size, 2277
- thr_suspend — suspend thread execution, 2278
- thr_yield — thread yield to another thread, 2280
- tigetflag — return the value of a terminfo capability, 2281
- tigetnum — return the value of a terminfo capability, 2281

tigetstr — return the value of a terminfo capability, 2281

time

- computes the difference between two calendar times — difftime, 534

time accounting

- for current process — times, 2289

time and date

- convert to string — asctime, 352
- convert user format date and time — getdate, 982
- get — ftime, 933
- settimeofday, 1088

time of day

- get and set — gettimeofday, settimeofday, 1086

time, calendar

- convert from a tm structure — mktime, 1440

timeout — set timed blocking or non-blocking read, 1567

timer_getoverrun — per-process timers, 2286

timer_gettime — per-process timers, 2286

timer_settime — per-process timers, 2286

times — get process times, 2289

tmpfile — create a temporary file, 2296

tmpnam — create a name for a temporary file, 2297

tnftcl_buffer_alloc — allocate or deallocate a buffer for trace data, 2299

tnftcl_buffer_dealloc — allocate or deallocate a buffer for trace data, 2299

tnftcl_check_libs — control probes of another process where caller provides /proc functionality, 2304

tnftcl_close — close a tnftcl handle, 2302

tnftcl_continue — interfaces for direct probe and process control for another process, 2310

tnftcl_exec_open — interfaces for direct probe and process control for another process, 2310

tnftcl_filter_list_add — control kernel tracing and process filtering, 2330

tnftcl_filter_list_delete — control kernel tracing and process filtering, 2330

tnftcl_filter_list_get — control kernel tracing and process filtering, 2330

tnftcl_filter_state_set — control kernel tracing and process filtering, 2330

tnftcl_indirect_open — control probes of another process where caller provides /proc functionality, 2304

tnftcl_internal_open — create handle for internal process probe control, 2307

tnftcl_kernel_open — create handle for kernel probe control, 2309

tnftcl_pid_open — interfaces for direct probe and process control for another process, 2310

tnftcl_probe_apply — iterate over probes, 2317

tnftcl_probe_apply_ids — iterate over probes, 2317

tnftcl_probe_connect — interfaces to query and to change the state of a probe, 2320

tnftcl_probe_disable — interfaces to query and to change the state of a probe, 2320

tnftcl_probe_disconnect_all — interfaces to query and to change the state of a probe, 2320

tnftcl_probe_enable — interfaces to query and to change the state of a probe, 2320

tnftcl_probe_state_get — interfaces to query and to change the state of a probe, 2320

tnftcl_probe_trace — interfaces to query and to change the state of a probe, 2320

tnftcl_probe_untrace — interfaces to query and to change the state of a probe, 2320

tnftcl_register_funcs — register callbacks for probe creation and destruction, 2325

tnftcl_strerror — map a tnftcl error code to a string, 2326

tnftcl_trace_attrs_get — get the trace attributes from a tnftcl handle, 2327

tnftl_trace_state_set — control kernel tracing
 and process filtering, 2330
 TNF_DEBUG — probe insertion interface, 2337
 TNF_PROBE — probe insertion interface
 arg_name_n, 2339
 arg_type_n, 2339
 arg_value_n, 2340
 detail, 2338
 keys, 2338
 name, 2337
 TNF_PROBE_0 — probe insertion
 interface, 2337
 TNF_PROBE_0_DEBUG — probe insertion
 interface, 2337
 TNF_PROBE_1 — probe insertion
 interface, 2337
 TNF_PROBE_1_DEBUG — probe insertion
 interface, 2337
 TNF_PROBE_2 — probe insertion
 interface, 2337
 TNF_PROBE_2_DEBUG — probe insertion
 interface, 2337
 TNF_PROBE_3 — probe insertion
 interface, 2337
 TNF_PROBE_3_DEBUG — probe insertion
 interface, 2337
 TNF_PROBE_4 — probe insertion
 interface, 2337
 TNF_PROBE_4_DEBUG — probe insertion
 interface, 2337
 TNF_PROBE_5 — probe insertion
 interface, 2337
 TNF_PROBE_5_DEBUG — probe insertion
 interface, 2337
 tnf_process_disable() — disables probing for
 the process, 2341
 tnf_process_enable() — enables probing for the
 process, 2341
 tnf_thread_disable() — disables probing for
 the calling thread, 2341
 tnf_thread_enable() — enables probing for the
 calling thread, 2341
 toascii — translate integer to a 7-bit ASCII
 character, 2343
 tolower — transliterate upper-case characters
 to lower-case, 2345
 _tolower — transliterate upper-case characters
 to lower-case, 2344
 touchline — control window refresh, 1175
 touchlock — functions to manage lockfile(s)
 for user's mailbox, 1355
 touchwin — control window refresh, 1175
 toupper — transliterate lower-case characters
 to upper-case, 2361
 _toupper — transliterate lower-case characters
 to upper-case, 2360
 towctrans — wide-character mapping, 2362
 tolower — transliterate upper-case
 wide-character code to
 lower-case, 2363
 toupper — transliterate lower-case
 wide-character code to
 upper-case, 2364
 tparm — return the value of a terminfo
 capability, 2281
 tputs — apply padding information and
 output string, 1774
 tracing — overview of routines for tnf tracing
 Tracing a Process, 2366
 Tracing the Kernel, 2367
 translate address to symbolic information —
 dladdr, 564
 translate integer to a 7-bit ASCII character —
 toascii, 2343
 translation between signal name and signal
 number — str2sig, 2092
 sig2str, 2092
 transliterate lower-case characters to
 upper-case — _toupper, 2360,
 2361
 transliterate lower-case wide-character code to
 upper-case — toupper, 2364
 transliterate upper-case characters to
 lower-case — _tolower, 2344,
 2345
 transliterate upper-case wide-character code to
 lower-case — tolower, 2363
 transport functions
 allocate memory, 2153
 truncate — set a file to a specified length, 2395
 tsearch — manage binary search trees, 2398
 ttyname — find pathname of a terminal, 2431
 POSIX, 2431
 ttyname_r — find pathname of a
 terminal, 2431

ttyslot — find the slot in the utmp file of the current user, 2433
twalk — manage binary search trees, 2398
typeahead — check for type-ahead characters, 2436
t_alloc — allocate memory for argument structures, 2153

U

uaddr2taddr — generic transport name-to-address translation, 1498
ualarm — schedule signal after interval in microseconds, 2437
ulltostr — string conversion routines, 2122
unctrl — convert character to printable form, 2438
ungetc — push byte back into input stream, 2439
ungetch — push character back onto the input queue, 2440
ungetwc — push wide-character code back into input stream, 2441
ungetwch — get (or push back) wchar_t characters from curses terminal keyboard, 426
unget_wch — push character back onto the input queue, 2440
unlock a pseudo-terminal master/slave pair — unlockpt, 2442
unlock a read-write lock object — pthread_rwlock_unlock, 1755
unlock address space — munlockall, 1444
unlock memory pages — munlock, 1442
unlockpt — unlock a pseudo-terminal master/slave pair, 2442
untouchwin — control window refresh, 1175
updwtmp — access utmpx file entry, 1096
updwtmpx — access utmpx file entry, 1096
use complex characters (and renditions) to draw borders — border_set, 265 — box_set, 265 — wborder_set, 265

use complex characters (and renditions) to draw lines — hline_set, 1122 — mvhline_set, 1122 — mvvline_set, 1122 — mvwhline_set, 1122 — mvwvline_set, 1122 — vline_set, 1122 — whline_set, 1122 — wvline_set, 1122
use single-byte characters (and renditions) to draw lines — hline, 1120 — mvhline, 1120 — mvvline, 1120 — mvwhline, 1120 — mvwvline, 1120 — vline, 1120 — whline, 1120 — wvline, 1120
user context — makecontext, 1357 — swapcontext, 1357
user IDs — get character-string representation — cuserid, 497
users — return information from remote machines — rusers, rnusers, 1927
use_env — set values of lines and columns, 2443
usleep — suspend execution for interval in microseconds, 2444
utmp file — access entry — getutent, 1093 — find the slot of current user — ttyslot, 2433
utmpname — access utmp file entry, 1093
utmpx file — access entry — getutxent, 1096
utmpxname — access utmpx file entry, 1096

V

variables used for X/Open Curses — global_variables, 1115
vfprintf — formatted output conversion, 1647
vfstab file

- getvfsent, 1100
- vfwprintf — wide-character formatted output of a stdarg argument list, 2445
- vidattr — display string with video attributes, 2446
- vidputs — display string with video attributes, 2446
- vid_attr — display string with video attributes, 2446
- vid_puts — display string with video attributes, 2446
- virtual memory
 - optimizing usage of user mapped memory
 - madvise, 1353
- vlfmt — display error message in standard format and pass to logging and monitoring services, 2447
- vline — use single-byte characters (and renditions) to draw lines, 1120
- vline_set — use complex characters (and renditions) to draw lines, 1122
- volmgt_acquire — reserve removable media device, 2449
- volmgt_check — have Volume Management check for media, 2452
- volmgt_feature_enabled — check whether specific Volume Management features are enabled, 2454
- volmgt_inuse — check whether or not Volume Management is managing a pathname, 2455
- volmgt_release — release removable media device reservation, 2458
- volmgt_root — return the Volume Management root directory, 2460
- volmgt_running — return whether or not Volume Management is running, 2461
- volmgt_symdev — convert between Volume Management symbolic names, and the devices that correspond to them, 2462
- volmgt_symname — convert between Volume Management symbolic names,

- and the devices that correspond to them, 2462
- vpfmt — display error message in standard format and pass to logging and monitoring services, 2465
- vprintf — formatted output conversion, 1647
- vsprintf — formatted output conversion, 1647
- vswprintf — wide-character formatted output of a stdarg argument list, 2445
- vsyslog() — log message with variable argument list, 2469
- VTOC, disk's
 - read a disk's VTOC — read_vtoc, 1794
 - write a disk's VTOC — write_vtoc, 1794
- vwprintf — wide-character formatted output of a stdarg argument list, 2445
- vwprintw — write formatted output to window, 1490
- vwscanw — read formatted input from window, 1492
- vw_printw — write formatted output to window, 1490
- vw_scanw — read formatted input from window, 1492

W

- waddch — add a character (with rendition) to a window, 182
- waddchnstr — copy a character string (with renditions) to a window, 184
- waddchstr — copy a character string (with renditions) to a window, 184
- waddnstr — add a multi-byte character string (without rendition) to a window, 186
- waddnwstr — add a wide-character string to a window, 188, 373
- waddstr — add a multi-byte character string (without rendition) to a window, 186
- waddwch — add a wchar_t character (with attributes) to a curses window and advance cursor, 368

waddwchnstr — add string of `wchar_t` characters (and attributes) to a curses window, 371
waddwchstr — add string of `wchar_t` characters (and attributes) to a curses window, 371
waddwstr — add a wide-character string to a window, 188, 373
wadd_wch — add a complex character (with rendition) to a window, 193
wadd_wchnstr — copy a string of complex characters (with renditions) to a window, 195
wadd_wchstr — copy a string of complex characters (with renditions) to a window, 195
wadjcurspos — moving the cursor by character, 374
wait — wait for process to terminate or stop, 2475
wait for and return LDAP operation result — `ldap_result`, 1291
wait for process to terminate or stop — `wait3`, 2471
 — `wait4`, 2471
wait for thread termination — `pthread_join`, 1721, 2262
wait for transmission of output — `tcdrain`, 2164
wait on a condition — `pthread_cond_timedwait`, 1704
 — `pthread_cond_wait`, 1704
wait3 — wait for process to terminate or stop, 2471
wait4 — wait for process to terminate or stop, 2471
watof — convert wide character string to double-precision number, 2490
watoi — convert wide character string to long integer, 2492
watol — convert wide character string to long integer, 2492
watoll — convert wide character string to long integer, 2492
wattroff — change foreground window attributes, 227, 376
wattron — change foreground window attributes, 227, 376
wattrset — change foreground window attributes, 227, 376
wattr_get — control window attributes, 225
wattr_off — control window attributes, 225
wattr_on — control window attributes, 225
wattr_set — control window attributes, 225
wbkgd — set the background character (and rendition) of window, 257
wbkgdset — set the background character (and rendition) of window, 257
wbkggrnd — set or get the background character (and rendition) of window using a complex character, 259
wbkggrndset — set or get the background character (and rendition) of window using a complex character, 259
wborder — add a single-byte border to a window, 261
wborder_set — use complex characters (and renditions) to draw borders, 265
wchar_t string
 number conversion — `wscanf`, 2521
wchgat — change the rendition of characters in a window, 294
wclear — clear a window, 301
wclrtoebot — clear to the end of a window, 308
wclrtoeol — clear to the end of a line, 309
wcolor_set — control window attributes, 225
wcrtomb — convert a wide-character code to a character (restartable), 2483
wcscat — wide-character string operations, 2499
wcschr — wide-character string operations, 2500
wscmp — wide-character string operations, 2499
wcscoll — wide character string comparison using collating information, 2485
wcscpy — wide-character string operations, 2500

wcsncpy — wide-character string operations, 2501
 wcsetno — get information on EUC codesets, 349
 wcsftime — convert date and time to wide character string, 2486
 wcslen — wide-character string operations, 2500
 wcsncat — wide-character string operations, 2499
 wcsncmp — wide-character string operations, 2499
 wcsncpy — wide-character string operations, 2500
 wcsprbrk — wide-character string operations, 2501
 wcsrchr — wide-character string operations, 2500
 wcsrtombs — convert a wide-character string to a character string (restartable), 2487
 wcsspn — wide-character string operations, 2501
 wcsstr — find a wide-character substring, 2489
 wcstod — convert wide character string to double-precision number, 2490
 wcstok — wide-character string operations, 2501
 wcstol — convert wide character string to long integer, 2492
 wcstombs — convert a wide-character string to a character string, 2495
 wcstoul — convert wide character string to unsigned long, 2496
 wcstring — wide-character string operations, 2499
 wcsvcs — wide-character string operations, 2501
 wcswidth — number of column positions of a wide-character string, 2503
 wcsxfrm — wide character string transformation, 2504
 wctob — wide-character to single-byte conversion, 2506
 wctomb — convert a wide-character code to a character, 2507
 wctrans — define character mapping, 2508
 wctype — define character class, 2509
 wcursyncup — synchronize window with its parents or children, 2132
 wcwidth — number of column positions of a wide-character code, 2510
 wdelch — remove a character, 510
 wdeleteln — remove a line, 513
 wechochar — add a single-byte character and refresh window, 635
 wechowchar — add a wchar_t character (with attributes) to a curses window and advance cursor, 368
 wecho_wchar — add a complex character and refresh window, 636
 werase — clear a window, 301
 wgetbkgrnd — set or get the background character (and rendition) of window using a complex character, 259
 wgetch — get a single-byte character from terminal, 975
 wgetnstr — get a multibyte character string from terminal, 1026
 wgetnwstr — get wchar_t character strings from curses terminal keyboard, 431
 wgetn_wstr — get a wide character string from terminal, 1028
 wgetstr — get a multibyte character string from terminal, 1026
 wgetwch — get (or push back) wchar_t characters from curses terminal keyboard, 426
 wgetwstr — get wchar_t character strings from curses terminal keyboard, 431
 wget_wch — get a wide character from terminal, 1103
 wget_wstr — get a wide character string from terminal, 1028
 whline — use single-byte characters (and renditions) to draw lines, 1120
 whline_set — use complex characters (and renditions) to draw lines, 1122
 wide character string to long integer, convert — watoi, 2492

- watol, 2492
- watoll, 2492
- wcstol, 2492
- wstol, 2492
- wide character string comparison using collating information
 - wcscoll, 2485
 - wscoll, 2485
- wide character string transformation
 - wcsxfrm, 2504
 - wsxfrm, 2504
- wide-character code classification functions
 - isenglish, 1180
 - isideogram, 1180
 - isnumber, 1180
 - isphonogram, 1180
 - isspecial, 1180
 - iswalnum, 1180
 - iswalpha, 1180
 - iswascii, 1180
 - iswcntrl, 1180
 - iswdigit, 1180
 - iswgraph, 1180
 - iswlower, 1180
 - iswprint, 1180
 - iswpunct, 1180
 - iswspace, 1180
 - iswupper, 1180
 - iswxdigit, 1180
- wide-character formatted output of a stdarg argument list
 - vfwprintf, 2445
 - vswprintf, 2445
 - vwprintf, 2445
- wide-character mapping — towctrans, 2362
- wide-character string operations
 - wscat, 2499
 - wcschr, 2500
 - wcscmp, 2499
 - wscpy, 2500
 - wcscspn, 2501
 - wcslen, 2500
 - wcsncat, 2499
 - wcsncmp, 2499
 - wcsncpy, 2500
 - wscprbrk, 2501
 - wcsrchr, 2500
 - wcsspn, 2501
 - wcstok, 2501
 - wcstring, 2499
 - wcsvcs, 2501
 - windex, 2500
 - wrindex, 2500
- wide-character to single-byte conversion — wctob, 2506
- winch — return a single-byte character (with rendition), 1141
- winchnstr — retrieve a single-byte character string (with rendition), 1142
- winchstr — retrieve a single-byte character string (with rendition), 1142
- windex — wide-character string operations, 2500
- winnstr — retrieve a multibyte character string (without rendition), 1153
- winnwstr — get a string of wchar_t characters from a curses window, 457, 1155
- winsch — insert a character, 1157
- winsdelln — insert/delete lines to/from the window, 1158
- winsertln — insert a line in a window, 1159
- winsnstr — insert a multibyte character string, 1160
- winsnwstr — insert wchar_t string before character under the cursor in a curses window, 451
- winsstr — insert a multibyte character string, 1160
- winstr — retrieve a multibyte character string (without rendition), 1153
- winswch — insert a wchar_t character before the character under the cursor in a curses window, 450
- winswstr — insert wchar_t string before character under the cursor in a curses window, 451
- wins_nwstr — insert a wide character string, 1162
- wins_wch — insert a complex character, 1165
- wins_wstr — insert a wide character string, 1162
- winwch — get a wchar_t character and its attributes from a curses window, 453

winwchnstr — get a string of `wchar_t` characters (and attributes) from a curses window, 455
 winwchstr — get a string of `wchar_t` characters (and attributes) from a curses window, 455
 winwstr — get a string of `wchar_t` characters from a curses window, 457, 1155
 win_wch — retrieve a complex character (with rendition), 1167
 win_wchnstr — retrieve complex character string (with rendition), 1168
 win_wchstr — retrieve complex character string (with rendition), 1168
 wmemchr — find a wide-character in memory, 2511
 wmemcmp — compare wide-characters in memory, 2512
 wmemcpy — copy wide-characters in memory, 2513
 wmemmove — copy wide-characters in memory with overlapping areas, 2514
 wmemset — set wide-characters in memory, 2515
 wmove — move cursor in window, 1449
 wmovenextch — moving the cursor by character, 374
 wmoveprevch — moving the cursor by character, 374
 wnoutrefresh — refresh windows and lines, 628
 wordexp — perform word expansions, 2516
 wordfree — perform word expansions, 2516
 working directory
 get pathname — `getwd`, 1106
 wprintf — print formatted wide-character output, 940
 wprintw — write formatted output to window, 1490
 wredrawln — redraw screen or portion of screen, 1814
 wrefresh — refresh windows and lines, 628
 wrindex — wide-character string operations, 2500
 write formatted output to window
 — `mvprintw`, 1490
 — `printw`, 1490
 — `vwprintw`, 1490
 — `wprintw`, 1490
 write screen contents to/from a file
 — `scr_dump`, 1959
 — `scr_init`, 1959
 — `scr_restore`, 1959
 — `scr_set`, 1959
 write_vtoc — read and write a disk's VTOC, 1794
 wscanf — convert formatted wide-character input, 947
 wscanw — read formatted input from window, 1492
 wscasecmp — Process Code string operations, 2522
 wscoll — Process Code string operations, 2522
 wscoll — wide character string comparison using collating information, 2485
 wscr — scroll a window, 1960
 wsdup — Process Code string operations, 2522
 wsetscreg — set terminal output controls, 302
 wsncasecmp — Process Code string operations, 2522
 wsprintf — formatted output conversion, 2520
 wsscanf — formatted input conversion, 2521
 wstandend — curses character and window attribute control routines, 376, 2086
 wstandout — curses character and window attribute control routines, 376, 2086
 wstod — convert wide character string to double-precision number, 2490
 wstol — convert wide character string to long integer, 2492
 wstostr — code conversion for Process Code and File Code, 2127
 wsxfrm — wide character string transformation, 2504
 wsynckdown — synchronize window with its parents or children, 2132

wsyncup — synchronize window with its
 parents or children, 2132
 wtimeout — set timed blocking or
 non-blocking read, 1567
 wtouchln — control window refresh, 1175
 wunctrl — convert a wide character to
 printable form, 2523
 wvline — use single-byte characters (and
 renditions) to draw
 lines, 1120
 wvline_set — use complex characters (and
 renditions) to draw
 lines, 1122

X

XDR library routines

- xdr, 2524
- xdrrec_endofrecord, 2527
- xdrrec_eof, 2527
- xdrrec_readbytes, 2527
- xdrrec_skiprecord, 2527
- xdr_admin, 2527
- xdr_control, 2527
- xdr_getpos, 2527
- xdr_inline, 2527
- xdr_setpos, 2527
- xdr_sizeof, 2527

XDR library routines for complex data structures

- xdr_array, 2530
- xdr_bytes, 2530
- xdr_complex, 2530
- xdr_opaque, 2530
- xdr_pointer, 2530
- xdr_reference, 2530
- xdr_string, 2530
- xdr_union, 2530
- xdr_vector, 2530
- xdr_wrapstring, 2530

XDR library routines for RPC

- rpc_xdr, 1923
- xdr_accepted_reply, 1923
- xdr_authsys_parms, 1923
- xdr_callhdr, 1923
- xdr_callmsg, 1923
- xdr_opaque_auth, 1923
- xdr_rejected_reply, 1923

- xdr_replymsg, 1923

XDR stream creation library routines

- xdrmem_create, 2533
- xdrrec_create, 2533
- xdrstdio_create, 2533
- xdr_create, 2533
- xdr_destroy, 2533

xdr_statstime — get performance data from
 remote kernel, 1925

xdr_statsvar — get performance data from
 remote kernel, 1925

xfn — overview of the XFN interface, 2535

XFN attribute

- fn_attribute_add, 769
- fn_attribute_assign, 769
- fn_attribute_copy, 769
- fn_attribute_create, 769
- fn_attribute_destroy, 769
- fn_attribute_first, 769
- fn_attribute_identifier, 769
- fn_attribute_next, 769
- fn_attribute_remove, 769
- fn_attribute_syntax, 769
- FN_attribute_t, 769
- fn_attribute_valuecount, 769

XFN attributes, a set of

- fn_attrset_add, 789
- fn_attrset_assign, 789
- fn_attrset_copy, 789
- fn_attrset_count, 789
- fn_attrset_create, 789
- fn_attrset_destroy, 789
- fn_attrset_first, 789
- fn_attrset_get, 789
- fn_attrset_next, 789
- fn_attrset_remove, 789
- FN_attrset_t, 789

XFN compound name

-
- fn_compound_name_append_comp, 799
- fn_compound_name_assign, 799
- fn_compound_name_copy, 799
- fn_compound_name_count, 799
- fn_compound_name_delete_all, 799
- fn_compound_name_delete_comp, 799
- fn_compound_name_destroy, 799
- fn_compound_name_first, 799

- - `fn_compound_name_from_syntax_attrs`, 799
 -
 - `fn_compound_name_get_syntax_attrs`, 799
 - `fn_compound_name_insert_comp`, 799
 - `fn_compound_name_is_empty`, 799
 - `fn_compound_name_is_equal`, 799
 - `fn_compound_name_is_prefix`, 799
 - `fn_compound_name_is_suffix`, 799
 - `fn_compound_name_last`, 799
 - `fn_compound_name_next`, 799
 - `fn_compound_name_prefix`, 799
 -
 - `fn_compound_name_prepend_comp`, 799
 - `fn_compound_name_prev`, 799
 - `fn_compound_name_suffix`, 799
 - `FN_compound_name_t`, 799
 - `fn_string_from_compound_name`, 799
 - an XFN context — `FN_ctx_t`, 829
 - an XFN identifier — `FN_identifier_t`, 832
 - XFN reference
 - `fn_ref_addrcount`, 839
 - `fn_ref_append_addr`, 839
 - `fn_ref_assign`, 839
 - `fn_ref_copy`, 839
 - `fn_ref_create`, 839
 - `fn_ref_create_link`, 839
 - `fn_ref_delete_addr`, 839
 - `fn_ref_delete_all`, 839
 - `fn_ref_description`, 839
 - `fn_ref_destroy`, 839
 - `fn_ref_first`, 839
 - `fn_ref_insert_addr`, 839
 - `fn_ref_is_link`, 839
 - `fn_ref_link_name`, 839
 - `fn_ref_next`, 839
 - `fn_ref_prepend_addr`, 839
 - `FN_ref_t`, 839
 - `fn_ref_type`, 839
 - XFN status object
 - `fn_status_advance_by_name`, 855
 -
 - `fn_status_append_remaining_name`, 855
 - `fn_status_append_resolved_name`, 855
 - `fn_status_assign`, 855
 - `fn_status_code`, 855
 - `fn_status_copy`, 855
 - `fn_status_create`, 855
 - `fn_status_description`, 855
 - `fn_status_destroy`, 855
 - `fn_status_diagnostic_message`, 855
 - `fn_status_is_success`, 855
 - `fn_status_link_code`, 855
 - `fn_status_link_diagnostic_message`, 855
 - `fn_status_link_remaining_name`, 855
 - `fn_status_link_resolved_name`, 855
 - `fn_status_link_resolved_ref`, 855
 - `fn_status_remaining_name`, 855
 - `fn_status_resolved_name`, 855
 - `fn_status_resolved_ref`, 855
 - `fn_status_set`, 855
 - `fn_status_set_code`, 855
 - `fn_status_set_diagnostic_message`, 855
 - `fn_status_set_link_code`, 855
 -
 - `fn_status_set_link_diagnostic_message`, 855
 -
 - `fn_status_set_link_remaining_name`, 855
 - `fn_status_set_link_resolved_name`, 855
 - `fn_status_set_link_resolved_ref`, 855
 - `fn_status_set_remaining_name`, 855
 - `fn_status_set_resolved_name`, 855
 - `fn_status_set_resolved_ref`, 855
 - `fn_status_set_success`, 855
 - `FN_status_t`, 855
 - `xfn_attributes` — an overview of XFN attribute operations, 2536
 - `xfn_compound_names` — XFN compound syntax: an overview of XFN model for compound name parsing, 2541
 - `xfn_status_codes` — descriptions of XFN status codes, 2548
 - XFN Status Codes, 2548
- ## Y
- `y0` — Bessel functions of the second kind, 2553
 - `y1` — Bessel functions of the second kind, 2553
 - `yield processor` — `sched_yield`, 1958
 - `yn` — Bessel functions of the second kind, 2553